(2)

Computer Graphics Research Laboratory
Quarterly Progress Report
No. 49

Norman I. Badler
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104-6389

Third Quarter 1993

November 22, 1993

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1993	Technical

4. TITLE AND SUBTITLE
Computer Graphics Research Laboratory
Quarterly Progress Report No. 49

5. FUNDING NUMBERS
DAAL03-89-C-0031

6. AUTHOR(S)
Dr. Norman I. Badler

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pennsylvania
Computer & Information Science Department
Philadelphia, PA 19104-6389

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARO 26779.21-MA-AI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during July through September 1993.

14. SUBJECT TERMS
Computer Graphics

15. NUMBER OF PAGES
84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

MEMORANDUM OF TRANSMITTAL

U.S. Army Research Office
ATTN: AMXRO-RT-IPL
P.O. Box 12211
Research Triangle Park, NC 27709-2211

____ Reprint (15 copies) XX Technical Report (50 copies)

____ Manuscript (1 copy) ____ Final Report (50 copies)

____ Thesis (1 copy)

____ MS ____ PhD ____ Other_____

CONTRACT/GRANT NUMBER DAAL03-89-C-0031

TITLE: Computer Graphics Research Laboratory Quarterly Progress Report No. 49

is forwarded for your information.

SUBMITTED FOR PUBLICATION TO (applicable only if report is manuscript):

Sincerely,

Dr. Norman I. Badler
University of Pennsylvania
Computer & Information Science Department
Philadelphia, PA 19104-6389

DAAL03-89-C-0031

Contents

1 Introduction: Norman I. Badler

This Quarterly Report includes descriptions of various projects underway in the Computer Graphics Research Lab during July through September 1993. These reports include:

- An overview of the new features of *Jack*®5.8.
- An update on the Naval Training Systems Center (NTSC) project to use *Jack* to control human figures within a distributed simulation environment.
- An introduction to the Parallel Transition Networks (PaT-Nets) package for designing and executing general object-oriented state-controllers for use with the motion-system or user-interaction with *Jack*.
- Additions to *Jack* 5.8 including CAD geometry converters and the Flock of Birds interface.
- An update on the new Collision Avoidance System in *Jack* which includes collision avoidance by the *Jack* human figure.
- A discussion of posture control in *Jack* including a set of basic postures and automatic transitioning between postures.
- Progress in locomotion for producing different styles of walking using torso flexion and pelvic rotation (TFPR).
- A description of the Object Specific Reasoner (OSR) prototype including its integration with high-level and search planners to create SodaJack, a system generating directives describing the motion of an animated agent.
- A brief description of the animation of the SodaJack system for SIGGRAPH.
- Plans for the creation of a system to generate conversational gestures.
- A discussion of human figure model enhancements supported by the Spreadsheet Anthropometry Scaling System (SASS).
- An update on X-SASS (SASS under X-Windows).
- Progress in the conversion of the Viewpoint Animation Engineering human model for use within *Jack*.
- The enhancement of Free Form Deformations (FFDs), including the incorporation of FFDs into the motion system, leading to the simulation of human internal organs.
- Progress in the animation of human reach trajectories using human reach data from the MOCO corporation and a *Jack* human model created using SASS.
- Enhancements to the motion system in the areas of interpolated motion events and extended motion events.
- Extensions to texture mapping to include the use of transparency in textures and texture filtering for ray-tracing.

- Plans for incorporating the *Jack* strength model with inverse kinematics for more realistic joint placement.
- Extensions of radiosity rendering.
- A discussion of blended shape primitives and the applications in computer vision and computer graphics.
- Advancements in facial animation, including the use of PaT-Nets for controlling the eye movements of agents involved in dialog.

There are also four appendices:

- *PaT-Nets:* Welton Becket. Documentation.
- *CVToJack Geometry Translation Program.* User documentation.
- *Hierarchical Shape Representation Using Locally Adaptive Finite Elements: A Model-Based Approach*: Eunyoung Koh, Dimitri Metaxas and Norman Badler.
- *A 3-D Model of Tongue Movements Using Soft Object Techniques*: Catherine Pelachaud, Chin Seah, C.W.A.M. van Overveld. To be sent to Computer Animation '94.

This research is partially supported by ARO DAAL03-89-C-0031 including U.S. Army Research Laboratory (Aberdeen), Natick Laboratory, and NASA Ames Research Center; U.S. Air Force DEPTH through Hughes Missile Systems F33615-91-C-0001; Naval Training Systems Center N61339-93-M-0843; Sandia Labs AG-6076; DMSO through the University of Iowa; NASA KSC NAG10-0122; MOCO, Inc.; Robotics Research Harvesting, Inc.; NSF IRI91-17110, CISE CDA88-22719, and Instrumentation and Laboratory Improvement Program #USE-9152503.

2 *Jack* 5.8: John Granieri

Our next general release of *Jack* will be *Jack* 5.8. 5.7 was essentially an internal lab version, and 5.8 will be the official release. I will ship it during the fourth quarter of 1993. Here's a quick overview of some of the new features that are new or revised since *Jack* 5.6 (the last external release of *Jack*):

New Human Body: *Jack* 5.8 has a new human body model. Your environments which contain human figures from 5.6 or earlier will need to be upgraded. The new human replaces the venerable human5 figure. Its most notable attributes are: more human looking; it has proper eyeballs; no more glasses and hat; better fingers; better feet; you can generate bodies directly from SASS; more geometry in the arms and legs to better approximate the human shape.

New CAD Translators: A set of new and enhanced CAD file translators are being released with *Jack* 5.8. In particular, there is a very good IGES translator, both in and out of *Jack*. There are also guidelines for building geometry in an external CAD system which will be viewed in *Jack*. There are also several new psurf utilities, which can reduce the node and face count in very complex geometry.

SASS Version 2.1: SASS has been in development for a while, and this is the first full external release of that program. It helps in creating human figures according to a variety of anthropometric variables.

Strength: The torque and strength computations for the human body are much improved. The data for available torque in the human body is from NASA Johnson Space Center. We have the available torque arm values now, and the leg values should be available shortly (this will only be an update to the body definition, all code is already within *Jack*).

Improved Animation System: The improvements to the animation system include: (1) motion groups, which allow you to group related motions together, and then create motion templates; (2) improved support within *Jack* for channels, which hold interpolated data, and are more extensible than the older frame structures; (3) motions and channels are now read and written using full Peabody syntax, not JCL. There are also several new motion types which make animating highly articulated figures much easier.

Articulated Hand: The articulated hand has been improved and the commands for using them are more robust. This will be followed by a more complete treatment of hand controls in the near term future.

LISP Programming Interface: *Jack* users who don't have access to source code can now extend *Jack's* functionality by writing Lisp code. A lisp interpreter is embedded in *Jack*, as well as lisp functions to access the internal structures of Peabody. This represents a very powerful way for users to extend *Jack*, and to share those extensions with others. We hope to distribute (in the near future) some sample lisp code which will demonstrate the power and flexibility of this new feature.

JCL Reference Manual: A preliminary version of a JCL Reference Manual will be released. It organizes the JCL commands alphabetically, and gives the full syntax for each command. This should make it easier to build JCL scripts, as well as provide a dense overview of *Jack's* functionality.

Utah Raster Toolkit 3.0: *Jack* now uses Version 3.0 of the URT. A subset of the URT is distributed with *Jack*. The URT is public domain software, and is included as a convenience to the *Jack* user.

Radiosity: An improved version of the radiosity renderer is included. This version uses a fast over-relaxation progressive refinement algorithm, and will export the finished rendered geometry back into *Jack*, keeping the psurfs independent (the older version would create one huge psurf out of an entire environment).

Improved Video Image Generation: *Jack* can now perform full-scene anti-aliasing and record 60 fields per second. This can dramatically improve image quality generated directly from *Jack* (when you're not using the ray-tracer).

Mirrors and Shadows: *Jack* 5.8 will include a preliminary release of real-time shadow and mirror generation. This should be very useful for analyzing visibility, as well as generating nicer images for presentation.

View and Clipping Planes: Support for user-definable clipping planes (up to 6) as well as commands for pushing and popping the view associated with a window. *Jack* also will push the view in a window when an automatic viewing adjustment is made, so you can then later pop back the previous view.

Rulers: You may create rulers which continuously read out the distance between two points in the environment. This is very useful for measuring fit and reach while placing or animating a human figure in a work environment.

Finer Grain Collision Detection: The collision detection routines have been greatly improved. They allow for tracking collisions between any two sets of segments in the environment, as well as choices for different intersection algorithms and display options.

Remote Commands: *Jack* can receive commands from remote processes through a *command port*. There are also a set of special JCL commands intended for use from remote processes. The Lisp interpreter can also communicate with external process through a *lisp port*.

Paths: You can now create an explicit path in *Jack*, which can then be used for animation or other uses.

Joint Motors: Joint motors are functions that allow the user to continuously "exercise" a degree of freedom for a particular joint.

Real-time Animation Preview: *Jack* can now preview your animations at close to 30 fps on an SGI workstation, using the SGI utility `movieplayer`.

Flock of Birds 6 DOF Sensor: Support for Ascension Technology's Flock of Birds 6 DOF Sensors has been added to *Jack*. You can use a bird to control the position and orientation of a figure or site in the *Jack* environment. We will also distribute the source code for the bird server process, so that you may easily modify it to support other types of sensors.

Contributed programs: There are several utility functions that are released with *Jack* as "contributed" commands. They aren't directly supported for now, but are released so that you can use them and give us feedback on their usefulness and applicability. Some of them include: (1) commands for using the GL fog parameters, to add fog to your windows, (2) fractal generation commands, (3) network commands for connecting two or more *Jack* processes with a shared environment, (4) particle system generators. There may be more by the time *Jack* 5.8 ships...whatever I can cram in before the deadline!

3 NTSC Project: John Granieri

We have made some progress on our work with Naval Training Systems Center. Essentially, they will be using *Jack* to control human figures within a distributed simulation environment. My work on this consists of:

- Building a method for updating human figures between a Peabody environment and a Performer environment. I use sockets between the two processes to send figure locations and joint transforms from *Jack* to Performer. This works fine for now, but we'll need to improve the connection for higher performance in the future. I've tried using shared memory, but am having technical difficulties so far. We're also creating loaders in Performer to load *Jack* databases (environments). Jonathan Crabtree is working with me on this part.

- The ability to request posture changes from the Performer side, and have *Jack* execute motions that create the desired posture change in the Peabody environment (See Section 8.) This seems to be a general need for interfacing *Jack* to other simulation systems: some external process pastes a set of motions on *Jack's* timeline, then requests *Jack* to execute the motions, updating the external simulation process as it goes.

- We will need lower resolution human figures, using simplified torso and hand segments for faster update on the Performer side, but which still allow *Jack* to treat the figure as a true "human" inside Peabody. This has implications on the *Jack* internal processes which manage the human figure, most notably the spine and behavior functions (which rely heavily on the human figure's articulation structure to build constraints to implement the behaviors).

- We will also need to change the interpolation of *Jack* motions from 30 fps to some lower value (say 10 fps), depending on the speed of the Performer process, and what frame rate it can achieve. This has broad implications on the structure of our motion system. I'm currently studying this now.

4 External Activities: John Granieri

- I attended the SGI Developer's Forum at Stanford in August and saw lots of interesting things that may end up being used in *Jack*. Some of the more interesting ones: (1) IRIS Performer - I'm benchmarking this now, and we may use it as the base for *Jack* (See above.) (2) Speech recognition - this might be useful for command input. (3) Distributed Shared Objects - this may provide a very good solution to extending *Jack* in the future (and alleviate the dreaded "link" time problems we have).

- I attended the SAE Human Modeling Technology Standards committee meeting at the Aerotech show in Costa Mesa. We had most of the developers of human modeling systems together, and we demo'ed our systems to each other. It was very enlightening. We hope to be very active in this committee, to help form the standards which should lead to much broader acceptance and use of the technology which *Jack* represents.

- I travelled to England to demonstrate *Jack* during the U.K./European launch of *Jack* by our distributor, GMS. We showed *Jack* in the Virtual Reality theater at the SGI booth at the CIM '93 show in Birmingham, as well as at British Aerospace (makers of Harrier Jump jet) and Vickers Defense (makers of the Challenger tank) as well as many others at the CIM show.

- I gave two two-day *Jack* Training Courses, in June and September. We had 10 attendees at each session, from various research sponsors and commercial users of *Jack*. You can get an attendee list from the UPenn Center for Technology Transfer.

5 Parallel Transition Networks: Welton Becket

The Parallel Transition Networks (PaT-Nets) package written in XLISP-STAT in *Jack*, allows users to design and execute general object-oriented state-controllers in *Jack* (See Appendix A for initial documentation.) It includes a set of macros for creating new networks and a simple net operating

system that time-slices running nets into *Jack* (so they run simultaneously with the motion-system or during user-interaction). A sample, abstract network is shown in Fig. 1.

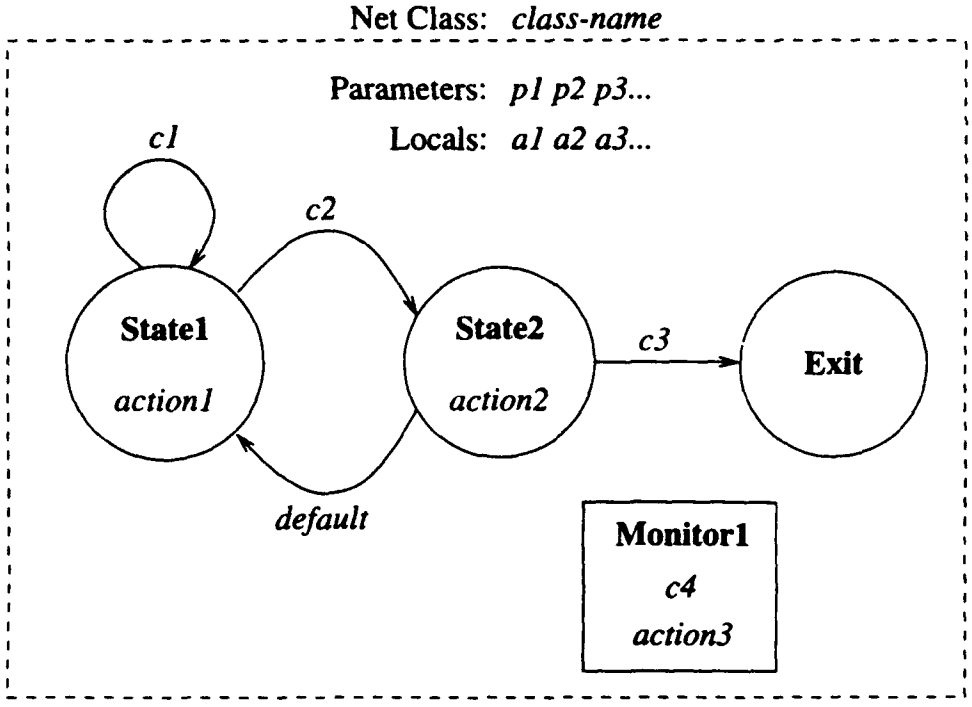


Figure 1: An example PaT-Net.

Each PaT-Net is a lisp class in the object-oriented sense (using the XLISP-STAT prototyping mechanism) whose primary components are a set of *nodes* and a set of *arcs*. Each node may contain an *action*, which may consist of arbitrary lisp code. Each arc has a *condition* containing an arbitrary lisp expression which should eval to true if the arc should be taken, and a target node defining the new state if the arc is taken. A PaT-Net may also contain an arbitrary number of *monitors*, consisting of a condition and an action – if at any time the condition is true the action is executed in the local space of the net. Networks are run by creating an instance of the PaT-Net class. Instances may take arguments, which allows parameterizing nets on creation. Each instance may have any number of local variables available to the actions and conditions. Since PaT-Nets are actually classes, new PaT-Nets can be made that inherit functionality from any number of parent PaT-Nets or that override or extend the functionality of parent PaT-Nets.

Actions and conditions may access all the *Jack*-API primitives in XLISP, and this allows them to become embedded *Jack* controllers. Actions can spawn new PaT-Net instances simply by instantiating new nets and the new nets will run in parallel with the parent. Actions can also execute *waits*, which take the running net out of the active list until some condition is met. A net can wait for:

- Another net to finish.
- Another net to pass through a specific node.

- A certain time (in the motion system).
- A semaphore or priority queue.
- An arbitrary lisp condition.

There are also facilities for creating probabilistic nodes that select arcs based on posted probabilities of the arc being chosen.

5.1 Planned extensions

- I plan to design and oversee the implementation of a graphical interface to constructing the PaT-Nets, which will be written in TCL. It will allow pasting manipulating nodes and arcs graphically and will have facilities for managing libraries of PaT-Nets.
- I am investigating the possibility of extending PaT-Nets to simulate Condition/Event Petri Nets by allowing multiple threads of execution in the same net and symbolic tokens that pass through the net.

6 New Additions in *Jack* 5.8: Mike Hollick

Most of the new developments I have mentioned in previous reports (e.g. rulers, real-time animation preview, user defined clipping planes) have been tested and included in *Jack* 5.8. Many bug fixes have been made, including one that will allow *Jack* to run over Distributed GL.

6.1 CAD Geometry Conversions

The *Jack* CAD Geometry Translator package is very near to completion. Over the past few months many bug-fixes have been made to existing translators, while much progress has been made on the new IGES translator. This IGES translator is very robust and will be the preferred tool for geometry conversion.

The ComputerVision translator is nearly complete. Preliminary documentation is included in this report as Appendix B.

I am in the process of checking and updating the other translators with *Jack* 5.8 to insure compatibility with the new release. All translators will be included in the 5.8 general release.

6.2 Flock of Birds Interface

The new Flock of Birds interface (as described in previous progress reports) will be included with *Jack* 5.8. The 4 sensor based human control module will also be included in the standard distribution. Extensive testing has led to several fixes to both the *Jack* side and the *flockd* control program.

Operation of a flock over one serial port is now supported. Currently, performance in this mode is not acceptable, but I am working on using hardware flow control on the line to significantly increase data throughput. This will allow any SGI machine (including Indigos and Indys) to be used with the Flock of Birds.

7 A New Collision Avoidance System: Xinmin Zhao

I have been re-designing/re-implementing the collision avoidance system to extend its functionalities, make it more robust, and hopefully more efficient.

Upper-body collision avoidance is being added. This will make many interesting collision avoidance behaviors possible (e.g., the one shown in Fig. 2, which was produced by the new collision avoidance system). In particular, we will be able to see the whole body coordination during collision avoidance.

In the previous implementation, there are 5 constraints on each hand for every obstacle segment being avoided. They are placed at the following positions: palm-center, wrist, upper lower-arm, elbow, and shoulder. The total number of constraints for each obstacle is 10. There are about 20 segments in the agent's body which have to be avoided (self collision avoidance), plus some environment obstacles. In a moderately complex environment, it is not uncommon to have more than 30 obstacles, which corresponds to more than 300 constraints in the system. With so many constraints, it is difficult to further improve the system performance.

In the new system, we adopt a different approach. Instead of having 10 constraints for each obstacle being avoided, we use 10 constraints, independent of the number of obstacles being avoided. These 10 constraints are shared by all the obstacles. As before, we have 5 constraints on each hand, and they are placed at the following joints: wrist, elbow, shoulder, waist, and knee. The first 2 constraints try to resolve collisions by moving the hand (arm), while the rest of the constraints try to avoid collisions by moving the whole body.

Even though there are fewer constraints in the new system, each constraint is doing much more work than before: now each of them has to avoid collisions with all the obstacles, not just one obstacle as before. So the performance gain will not be as significant as the reduction of the number of constraints.

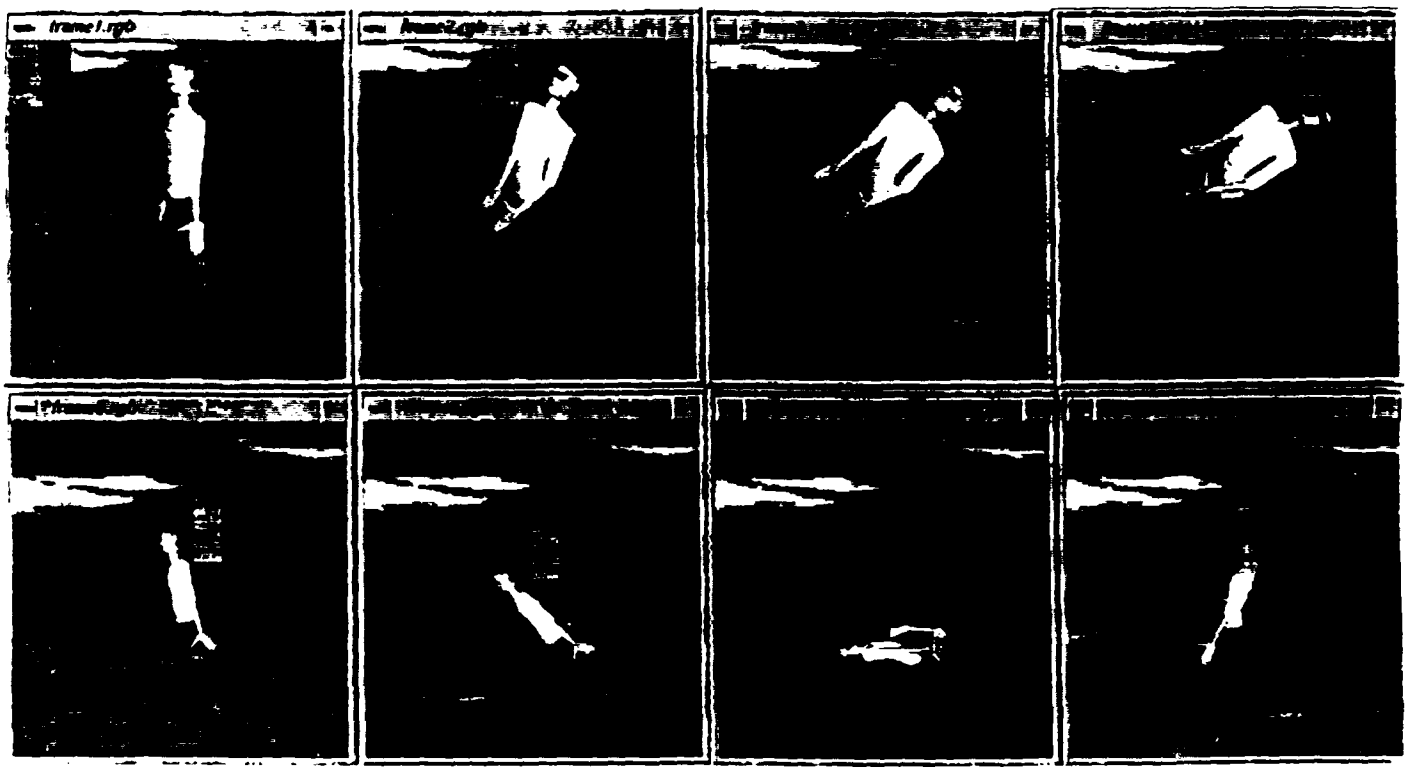


Figure 2: *Jack* avoids collision with an overhead obstacle.

8 Posture Control Network: Ramamani Bindiganavale, Kok-Hoon Teo, Susanna Wei

One of the biggest challenges most *Jack* users have is to get the human figures into the right positions. Though a set of manipulation procedures is available for this purpose, it still requires much time and effort from the user to get it right. For each newly created human figure, the process of setting it in the right posture has to be repeated. It is thus clear that the current set of primitives for direct

manipulation of human figures is not sufficient for simulations and animations involving more than just a few human figures.

The main goal of this project is to introduce the notion of posture into *Jack*. Posture is the arrangement of bodily parts within the human figure. However, we would also like to extend this notion to all computer generated articulated figures in general. The main objectives here are to:

1. Implement a predefined set of human postures in *Jack* so that the user can simply point and click to fix figures in a selected posture.
2. Implement a posture control network enabling smooth transitions between any two arbitrary postures.

The Posture Network consists of a set of static postures and a finite state machine controlling the transitions between them. From the perspective of a *Jack* user who performs human factor simulations, it is essential to jump start human figures into a desired posture. With the current set of human control and manipulation primitives, a user needs to understand how the behavior system works so that the human figure can be set to the desired posture.

A number of basic static postures (stand, squat, sit, supine, prone, kneel, etc.) have been generated. The user, without an in-depth understanding of the behavior system, can now get the human figure directly into any one of these postures. These postures can be further used to build more complicated postures or to move the human figure from one posture to another posture.

Motion sequences have been built to enable transitions between adjacent sets of postures. A finite state machine has been built to keep track of the figure postures and to generate the set of required motion sequences to transit between any two arbitrary postures. Thus the user can generate animations of motion between any two postures without directly manipulating the human figure. Fig. 3 illustrates the transition graph depicting the relationship between the postures used in the finite-state system.

The transition graph is represented as a matrix in Fig. 4. The problem of finding the required motion sequence for transition between any two given postures corresponds to finding a path connecting them in the transition graph. The algorithm to generate the complete transition sequence is as follows:

Transit (x:posture,y:posture,M:Matrix)

1. If M(x,y) = -1 then stop.
2. If M(x,y) contains a motion sequence (denoted by lower-case characters), then execute the sequence and stop.
3. If M(x,y) contains posture x, then execute M(x,z) followed by M(z,y)

Some of the predefined sets of postures have been illustrated in Fig. 5, 6, 7, 8.

The current postures have all been defined solely in terms of joint angles. The main disadvantage of this method is that the same set of joint angles applied to an anthropometrically scaled human

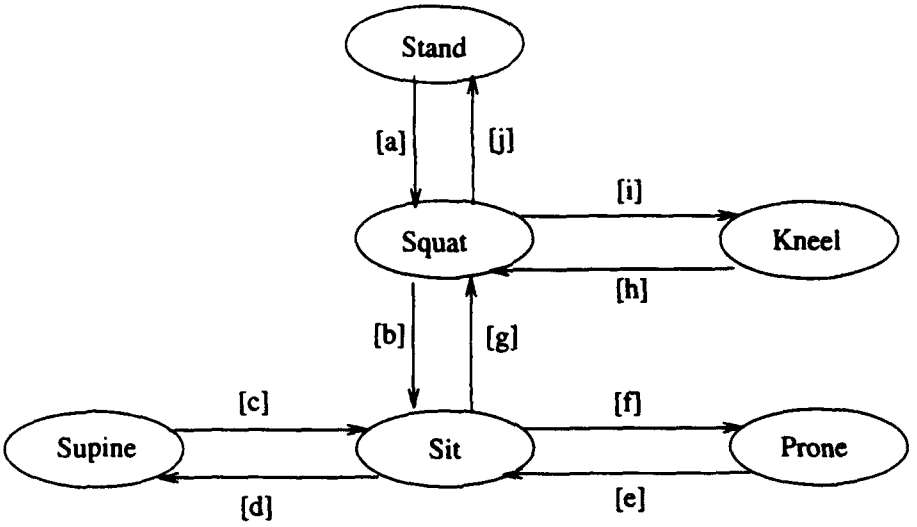


Figure 3: Posture Transition Graph.

Final / Initial	0	1	2	3	4	5
0 (Stand)	-1	a	1	1	1	1
1 (Squat)	j	-1	3	b	3	h
2 (Supine)	3	3	-1	d	3	3
3 (Sit)	1	g	c	-1	e	1
4 (Prone)	3	3	3	f	-1	3
5 (Kneel)	1	i	1	1	1	-1

Figure 4: Posture Transition Matrix.

figure does not guarantee a visually similar appearance. Further research has to be done to generalize the basic static postures and the intermediate transitions to any anthropometrically sized figure. Many more key postures and the transitions between them have to be identified and added to the database of postures to reproduce most human actions.

As part of the NTSC project, a new posture system (new static postures and transitions) has to be built. One of the basic postures is shown in Fig. 9.

Figure 5: Static Squatting Posture

Figure 6: Static Sitting Posture

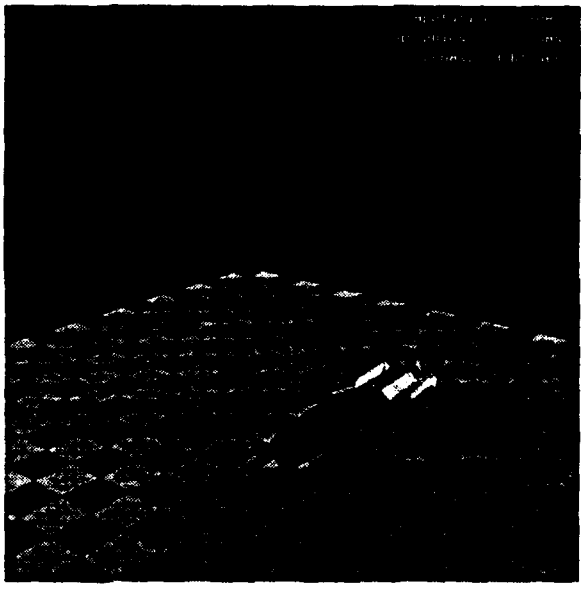


Figure 7: Static Supine Posture

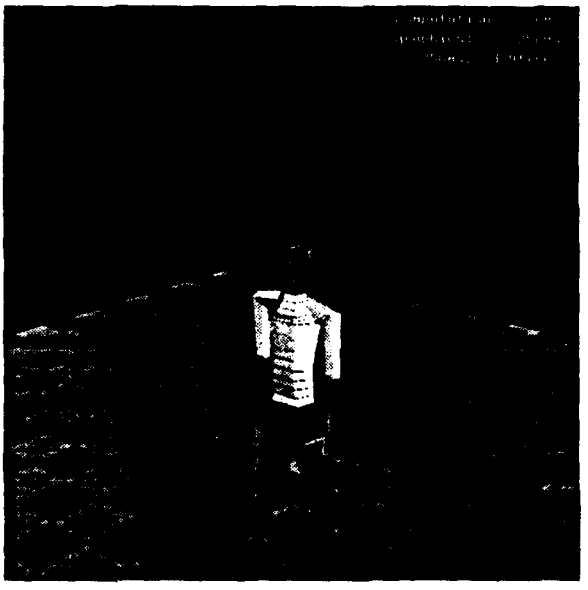


Figure 8: Static Kneeling Posture

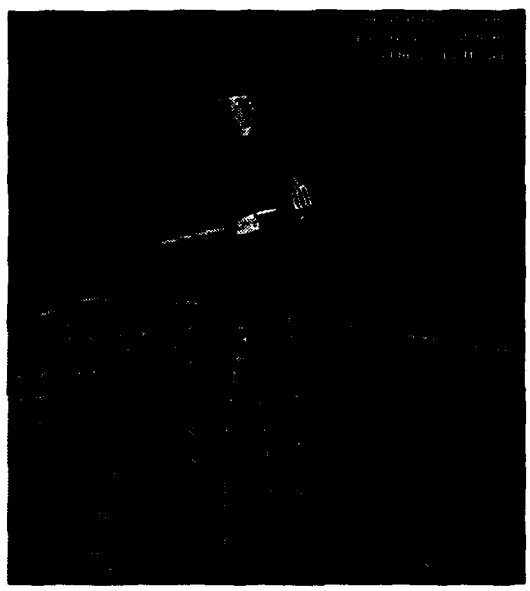


Figure 9: Soldier Holding a Rifle

9 Stylistic Walking with Flexible Torso and Pelvic Rotations: Hyeongseok Ko

For the locomotion animation in *Jack*, a biomechanical measurement of straight line walking is generalized to the motion of an arbitrary anthropometrically scaled human in stepping along any curved path, including intermittent non-rhythmic stepping such as turnaround or lateral stepping.

The torso flexion and pelvic rotation (TFPR) during walking had been determined from the original measurements. In the previous quarter, the TFPR has been parameterized to generate different styles of walking. Sinusoidal functions are used: two parameters control the shape of the curve, i.e., the amplitude and displacement.

Twelve parameters are used to specify the pelvic rotation pattern: A_θ^x, D_θ^x, A_θ^y, D_θ^y, A_θ^z, D_θ^z, A_p^x, D_p^x, A_p^y, D_p^y, A_p^z, D_p^z. The first six values are for angular rotation, and the latter six are for the translation of the pelvis compared to the normal walking step. The x axis is along the forward direction, y axis is lateral (rightward), and z axis is downward. With these parameters the actual pelvis rotation at the normalized time t is given as

$$\theta_x(t) = A_\theta^x sin(2\pi t) + D_\theta^x. \quad (1)$$

The other rotations or translations are described similarly except that they may be cosine functions instead of sines.

For example, if we set $A_\theta^z = 10$, we can observe a rhythmic pelvis rotation. By setting $D_p^z = 20$, the center of mass is lowered $20cm$ uniformly during the step. In addition to that, if we set $D_{\theta'}^y = 20$ and $D_p^x = -10$, the torso is bent forward by 20 degrees and the hip is placed $10cm$ backward, resulting in crouched walking.

The flexion of torso is controlled similarly by sinusoidal functions with the parameters: $A_{\theta'}^x$, $D_{\theta'}^x$, $A_{\theta'}^y$, $D_{\theta'}^y$, $A_{\theta'}^z$, $D_{\theta'}^z$.

10 Prototyped OSR in LISP: Libby Levison

A prototype of the Object Specific Reasoner (OSR) has been written in Lucid Common Lisp. The OSR is the intermediate planning level of the AnimNL and SodaJack systems, and is responsible for tailoring general action plans to the specific agent, object and situation of their use.

Prototyping the OSR entailed specifying a small knowledge base with supporting access and retrieval functionality and building representations of all the objects in the scene. A small set of task-action procedures were defined, and the task-action elaboration mechanism was coded. The output of the OSR is currently a list of motion commands written to a file; full integration to the motion control system is expected by the end of 1993.

With the basic structure of the OSR in place, the OSR was integrated with Chris Geib's high-level planner "ItPlanS", and Mike Moore's search planner "SearchPlan" to build the SodaJack system. SodaJack takes menu-commands such as "serve soda" or "serve icecream" and results in

a list of motion commands for the animated agent to perform. A rough sketch of the system is: ItPlanS receives the menu command, and identifies a plan which will accomplish the stated goal. All references to objects are resolved at this level when ItPlanS calls SearchPlans to bind referents to objects in the world. The plan is then expanded into steps or task-actions. Each task-action is sent to the OSR, and the OSR confirms that the agent can perform the requested action on the object (or not). ItPlanS uses this information to select the plan expansion to follow. Once ItPlanS commits to a task-action, it is sent to the OSR and the OSR generates the set of motions for the agent to perform. These are written to a *Jack* Command Language file, and read into the *Jack* system separately.

In the case that the action cannot be performed, it is often the case that a tool could be used to license the action. The OSR generates a description of the attributes such a tool would need, and returns this to ItPlanS. If ItPlanS so chooses, it can invoke SearchPlans and request a search for such a tool. SearchPlans requires access into the OSR knowledge base for this purpose.

Specifying the interfaces between the OSR and ItPlanS, and that between the OSR and SearchPlans took most of a month. A parser was written to interpret instructions from ItPlanS, and to retrieve and add information required by the OSR but missing from the ItPlanS command. SearchPlans also needs access to OSR specific knowledge; additional access routines into the OSR knowledge base for SearchPlans.

The final system generates lists of motion directives describing the motions of the animated agent. The next step is to port the code to XLISP, and then to integrate directly with the *Jack* system.

11 SIGGRAPH Movie: Libby Levison

As a test bed for the integrated planning system, we created animations demonstrating the probable outcome of running SodaJack. The animations were generated by Brian Stokes, under my supervision. We designed the presentation format of SodaJack and built the narrative slides that describe the system and its functionality. This requires illustrating the integration and flow of control between the three modules in the SodaJack system. With Len Wilson, I edited together the final version of the SodaJack 11 minute movie.

12 Gestures: Brett Achorn

This quarter I will be working with Dr. Justine Cassell to develop a system to generate conversational arm and hand gestures. Some of the computational issues to be addressed will include identifying gesture primitives; instantiating gestures based on intention, emphasis, and timing; and developing heuristics for correctly transitioning from one gesture to the next without drifting out of the comfort range for an average person or interfering with subsequent gestures.

13 SASS Update: Francisco Azuola

During the past quarter, SASS v.2.0 has been under test. As a result, the release version 2.1.3 has come out as the one chosen to be part of *Jack* 5.8.

In essence, v.2.0 and up provide direct connection between SASS and *Jack*. Important features regarding v.2.1.3 are:

1. The database code has been cleaned up to provide a more robust and flexible access to this facility. Some bugs and suggestions were pointed out by users of v.2.0, that have been corrected in v.2.1.3.

2. User feedback has resulted in extensive debugging and revision of the program. A major revision has been done regarding the figure creation. This is perhaps the most important accomplishment. When SASS v.2.0 was released, users pointed out that the human body model, namely polybody, was quite unrealistic. Up to this point, the major issue in SASS had been to preserve accuracy along the scaling process. But now we have entered into a new phase of our design, in which the users demand "good looks" of the model, i.e., the polybody was criticized for not being appealing enough. In order to provide a temporary solution to this problem, new geometry was developed. SASS v.2.1.3 has been updated to support this new geometry.

3. Regarding the new geometry, the major changes made are:

   - Increase the number of faces per segment, for all the limbs, to allow for better shaping.
   - Hip and buttocks have been given a better looking shape.
   - Improving the overlap between segments has been especially considered. In particular, new torso segment overlaps have decreased the formation of gaps considerably.
   - The scaling process for the torso itself has been refined.
   - A new head has been modeled, including eyes, eliminating the need for the sunglasses that were used in the past due to the lack of eyes.
   - The sitting posture of the figure has been noticeably improved, by a better positioning of hip joints.

4. In general, the new geometry has also allowed for a more accurate scaling process, and has pointed out some problems in this process that will be addressed in a future version of SASS.

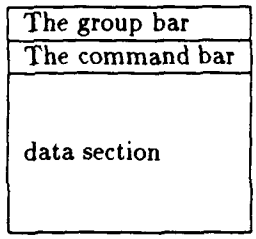
5. Work is currently being done regarding geometry, as we recognize there is still room for improvement. The polybody model needs to be refined a lot more. Furthermore, the introduction of the Viewpoint Animation Engineering geometry for the human body will provide a high quality human model in the future, at the expense of increased complexity.

6. Work also is being done on the interface for SASS. To allow for better portability, and to improve the user interface with the program, a new one is being designed and developed using TCL environment, which is going to be introduced in *Jack* as well (See next section.)

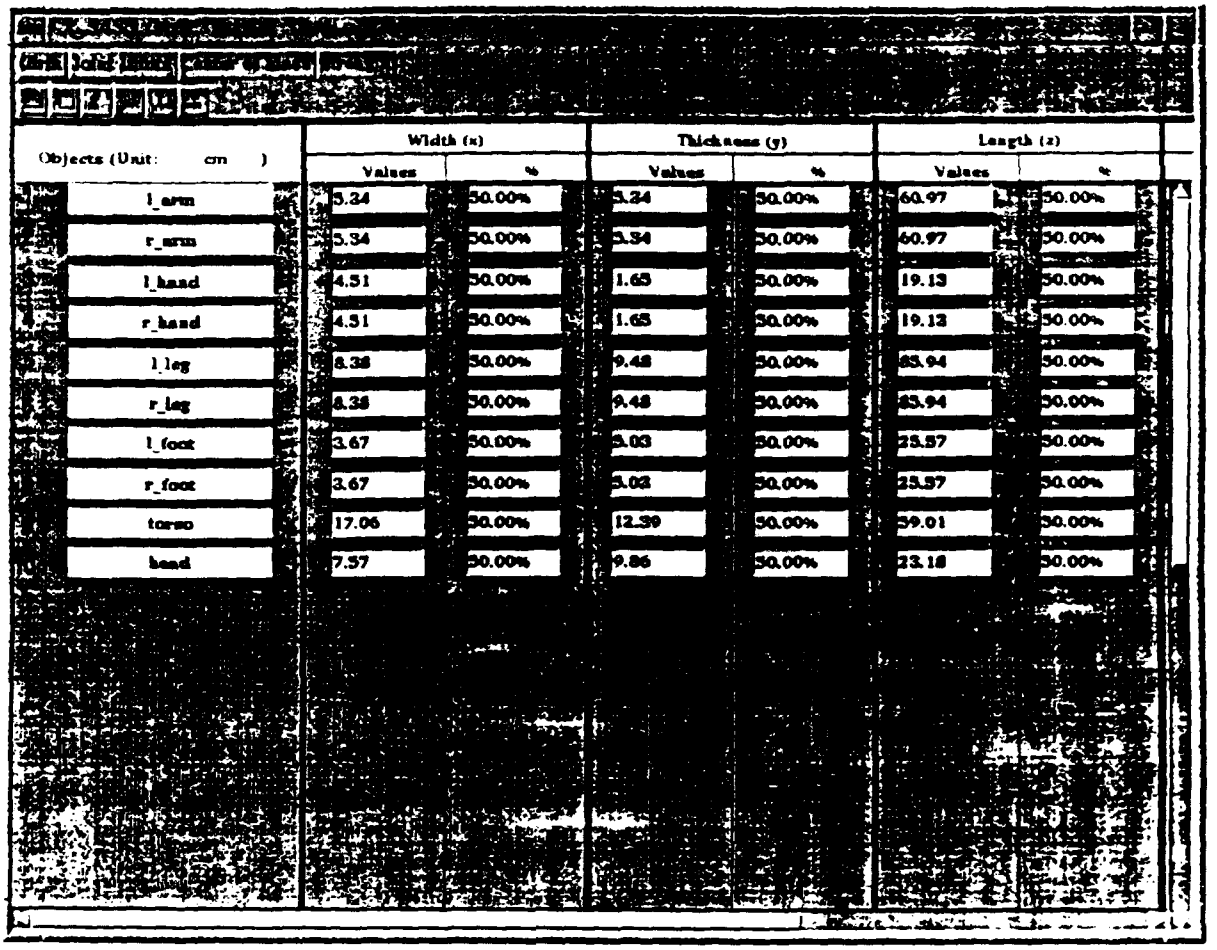
14 X-SASS: Ann Song, Francisco Azuola, Susanna Wei

In order to make it easier for programmers to maintain their code and for users to extend their applications, we are in the process of porting SASS to X windows using TK. This new version of SASS running under X windows is called X-SASS.

To achieve a user-friendly interface in X-SASS, we use picture buttons, dialogue boxes, and pull-down menus. Users can resize the window and scroll up/down and left/right on the spreadsheet of X-SASS. They can also work on multiple windows. The X-SASS screen layout is as follows:



Figures 10 and 11 are sample spreadsheets designed for X-SASS. Since TK is a C library package, we will write C programs to communicate between TK and the current SASS.



Objects (Unit: cm)	Width (x)		Thickness (y)		Length (z)	
	Values	%	Values	%	Values	%
l_arm	5.24	50.00%	5.24	50.00%	60.97	50.00%
r_arm	5.34	50.00%	5.34	50.00%	60.97	50.00%
l_hand	4.51	50.00%	1.65	50.00%	19.13	50.00%
r_hand	4.51	50.00%	1.65	50.00%	19.12	50.00%
l_leg	8.38	50.00%	9.48	50.00%	85.94	50.00%
r_leg	8.38	50.00%	9.48	50.00%	85.94	50.00%
l_foot	3.67	50.00%	5.03	50.00%	25.57	50.00%
r_foot	3.67	50.00%	5.03	50.00%	25.57	50.00%
torso	17.06	50.00%	12.39	50.00%	59.01	50.00%
head	7.57	50.00%	9.86	50.00%	23.18	50.00%

Figure 10. Sample display of X-SASS.

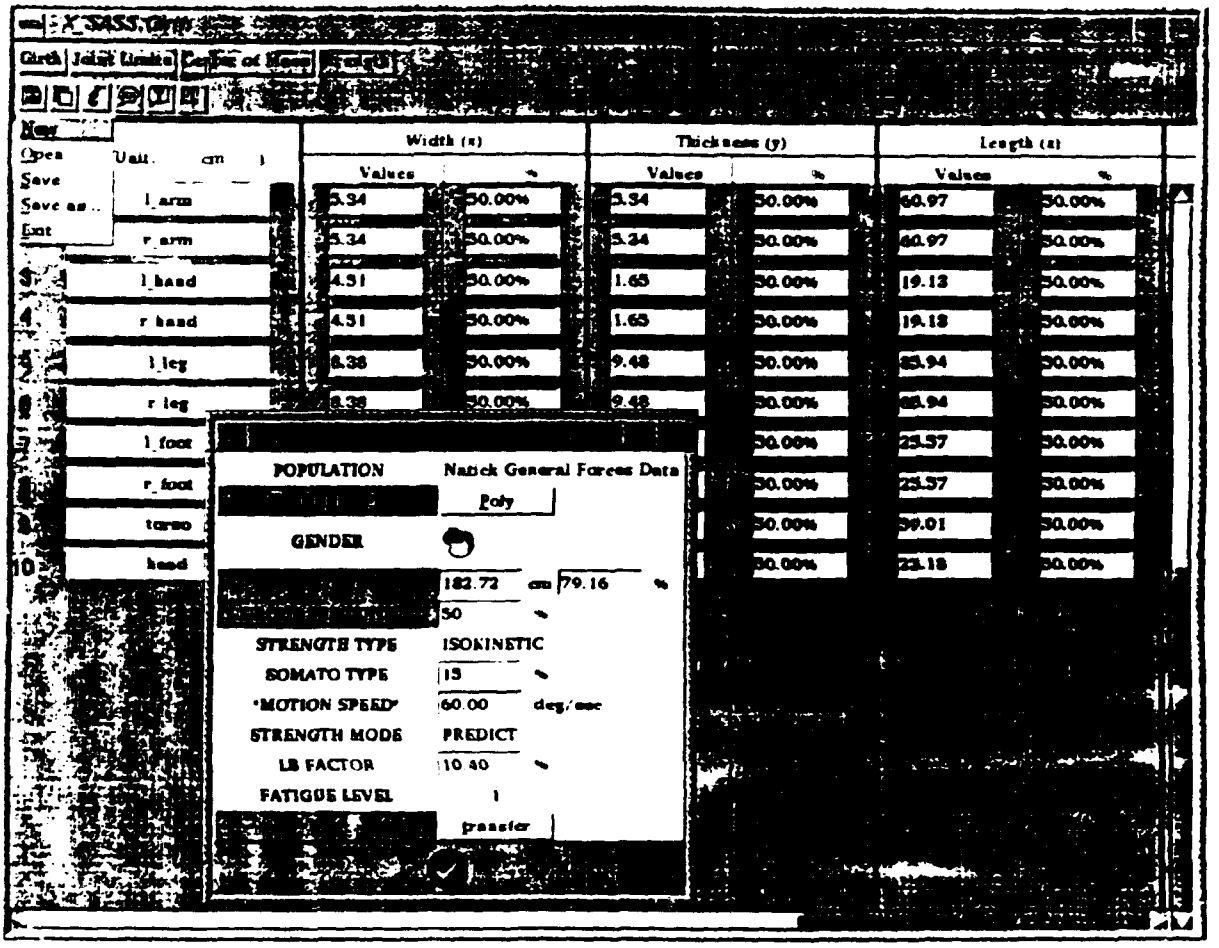


Figure 11. Sample of the command bar: Input parameters

15 Viewpoint to *Jack* Conversion: Pei-Hwa Ho

We have converted the human model from Viewpoint Animation Engineering to the standard *Jack* format.
The conversion involves:

- Transforming the segment geometry to the proper coordinate frame.
- Define the proper joint centers on the segment.
- Slicing the upper torso to conform to the seventeen segment torso model.

We have also developed the program that switches between the standard body and a Viewpoint body. This will be useful since the standard body has a lot fewer polygons than the Viewpoint body. We are waiting for the detailed hand model from Viewpoint in order to created a comparable hand model.

To make the Viewpoint model more useful, we need to:

- Normalize the segment geometry in order to create models of different sizes.
- Decimate the detailed geometries to create a model of lesser detail but easier to manipulate in real time.

16 Free-Form Deformation (FFD): Bond-Jay Ting

In this quarter, multiple attachment between deformation lattice and segments has been achieved. Also the preservation of joint positions of the human spine is completed.

- **Preservation of Joint Positions:** One of the characteristics of a spline curve is smoothing. When a control node position is changed, other control nodes will smooth out the applied object and reduce the effect from that control node. This smoothing characteristic also makes the exact node position very hard to predict. When applying free-form deformation to the human body, this smoothing characteristic will push joints away from their positions. In this quarter, I was working on correcting this problem and preserving the joint positions. That is, the joint positions from deformation lattice motions should coincide with the spine joint positions from articulated rigid body motions.
- **Multiple Attachment:** With this effort, one single deformation lattice can be attached to more than one segment. This not only speeds up the computation time, but also makes possible object selected deformation in the same 3-D space. Inside the same space, there may be an existing deformable object as well as an object which is not deformable. With this effort, we can apply deformation on deformable objects and use rigid body motion on those objects which

are not deformable. One implementation is simulating human torso movement with human organs. Inside the human torso, there are human organs which consist of soft tissues which are deformable. But there also exists bone structure which should be treated as rigid body. With this attachment selection, we can simulate it with both rigid body motion and free form deformation.

In this quarter, I also added some animation functions which make deformation animation possible.

Next quarter, I will implement the inner organ simulation by using both rigid body motion and free form deformation. I will also concentrate on adding material properties into deformations.

17 Human Reach Trajectory Animation: Hanns-Oskar Porr

The MOCO corporation has supplied us with a large number of trials of human reach data. In this stage of my research it is my task to visualize this data.

In the experiments conducted by MOCO, a human subject reached for several predetermined points directly ahead and/or above. Located on the human's torso and arm, 4 Ascension Technology Bird magnetic field sensors were placed. Each sensor can measure its location and orientation (6 DOF) in reference to a sampling device. Thus, this setup enabled MOCO to digitally sample the resulting reaching motions.

In the first step in creating the animation, I used the SASS system developed by Francisco Azoula to model a synthetic actor that fits the anatomy descriptions given by MOCO, e.g. length of forearm, clavicle, etc. This standard human *Jack* model then had to be augmented with a improved wrist and forearm model developed by Jianmin Zhao for his Ph.D. research. This new wrist enables the figure to twist the forearm more realistically, and thus is a better fit for the simulation. Next, four new sites were placed on the torso and arm that model the location and orientation of the Bird sensors, in accordance with the human landmark measurements supplied by MOCO.

The actual animation is accomplished using the constraint solver in *Jack*. I created four constraints, that restrain the aforementioned sensor sites to 4 goal sites. These goal sites are moved from one frame to the next to the location/orientation of the measured data as supplied by MOCO. Whenever this happens, the internal constraint solver tries to find a body configuration that puts the sensor sites into a position that matches these goal site – it tries to solve the constraints of the system.

Initial results of the motion look very good. The constraint solver seems to find or approximate the desired position/orientation very well. There are some minor adjustments that will have to be made with reference to the measurements. For instance the landmark measurements of the sensor placements were taken in reference to one of the edges of the physical sensor, and not to its center where the actual magnetic measurement occurs. Thus, the site placement is slightly off.

One other problem that I will have to solve in the near future is that the constraint solver is too slow for real time recording. I will have to write an extension to the system that collects the frames

and plays them back at a faster rate.

18 Additions to Motion System: Paul Diefenbach

Interpolated spline path motion was extended to facilitate camera motion. Path point manipulation and creation were modified to permit view attachment. Quaternion interpolation was extended to permit individual path segment orientation reversal (i.e. clockwise or counterclockwise interpolation).

Jack version 5.7 now includes two commands to facilitate extended motion events. The first is a Timed JCL command to place any JCL command on the timeline. The second is a Go and Record command to record each frame as the Go is being performed. This allows for animating previously unrecordable events such as color changes and deformations.

I worked with Ben Ting to incorporate FFDs into the motion system. Two commands to create FFD motions were added. Used in conjuncture with the Go and Record command I added, as well as the Timed JCL command, deformations are now animateable.

19 Textures and Transparency: Paul Diefenbach

Texture mapping was extended to include the use of alpha values (transparency) in textures. Two types of texture transparency are supported; namely alpha-blend and alpha-only. Alpha-blend uses each pixel's alpha value to blend its color; alpha-only draws with full translucency all pixels whose alpha value is not equal to zero. Alpha-only uses Z-buffering; alpha blend does not.

Alpha-only and untextured transparent objects are now drawn in a sorted order after non-transparent objects based on their rootsite's distance from the viewer. This permits more accurate transparency than previously allowed.

20 Texture Sampling and Strength Guided Motion: Jeffry S. Nimeroff

During the third quarter of 1993, I completed my work on the Convolution Mask Approximation Module (fast texture filtering while ray-tracing) and sent the work off to Computer Graphics Forum.

I have recently started working on incorporating the strength model in *Jack* into the inverse kinematics algorithm for end-effector positioning (for Siggraph '94). The strength data is used to prune the search space of valid joint angle positions and we hope to achieve more realistic joint placement than without using the model.

21 Radiosity: Min-Zhi Shao

In the radiosity program coding part I finished a radiosity environment decomposition program for rendering complex environments. I also revisited the whole radiosity package, polished the code, and made some minor changes of the user interface.

The current radiosity system consists about 12,000 lines of C code. Its major functions include:

- interactive and automatic meshing in environment modeling;
- fast overrelaxation progressive refinement algorithm;
- adaptive environment subdivision in radiosity calculation;
- texture mapping;
- output decomposition for walking through complex environments.

My major future research is to develop and implement more efficient algorithms and techniques for rendering dynamic and complex environments.

In mathematics, the radiosity equation is the Fredholm integral equation of the second kind [9]. Conventional numerical approaches to solve the Fredholm integral equation are mainly the projection methods such as collocation method and Galerkin method [4][5]. In the past ten years, with the rapid growth of computing power, several new numerical methods have been developed. Among them, to my knowledge, multigrid method [11] and wavelet method [7], have already been successfully applied to solve the Fredholm equation ([10][11] for multigrid and [1][2][3] for wavelet).

Multigrid method is a fast solution technique for finite difference and finite element approximations of differential or integral equations [10][11]. The main idea is to solve a problem on a succession of scales, propagating low spatial frequency from coarse grids to fine grids, and propagating high spatial frequency from fine grids back to coarse grids. To my knowledge, general multigrid method has not yet been applied to the radiosity solution.

Alpert et al [1][2][3] noticed that in many cases of interests after decomposing the integral equations, while the matrices involved are dense, their elements vary smoothly as a function of their indices, except along a collection of bands of fixed width. Then, their major idea was to use the wavelet to transform such locally smooth matrices into sparse matrices which support fast algorithms for matrix application and inversion. Gortler et al [8][13] used Alpert et al's results to ideal diffuse radiosity solution.

Recently, Briggs and Henson [6] presented a short note in which the authors explored the suggestive similarities between multiresolution wavelet and multigrid approaches to general operator equations.

It should be noted that in all above numerical methods, the kernel of the Fredholm integral equation of the second kind is assumed to be given. Apparently, this is not the case in radiosity

solution while the visibility calculation usually costs more than 90% of the total computation. However, I suspect the wavelet solution can be more advantageous than the conventional method for general non-diffuse environment. Two points: 1) The matrix equation becomes huge with directional radiosity. Immel et al [12] once reported a test result with 8 hours of visibility calculation versus 192 hours of matrix solution for a simple environment. The Alpert et al's algorithms might be more advantageous in this case. 2) One of the latest results of the conventional method is to use spherical harmonic decomposition to represent the directional surface radiosity distribution [14]. It should be quite obvious that, in general, the directional radiosity distribution is non-stationary. But, using compact support multiresolution bases to represent non-stationary signal is the major consideration to choose wavelet transform instead of the conventional Fourier transform. Besides the wavelet method, I think it should also be interesting to investigate the possibility to use multigrid techniques to accelerate radiosity solution.

It was until recently that I began to investigate the literatures of new developments in the second kind equation. Though my understanding of the subject is still rather superficial, I found the new thoughts quite fascinating and tended to believe it might well lead to new results in radiosity illumination solution.

22 Blended Shape Primitives: Douglas DeCarlo

The physics-based modeling paradigm used and developed by Dimitri Metaxas uses parameterized primitives (e.g. superquadrics) to represent basic shapes. More complex shapes are attained by using global deformations such as tapering and bending which are defined independently of the underlying parameterized primitive. Another way of introducing more complex shapes is to use other parameterized shapes such as supertoroids.

One drawback to using parameterized shapes is that it is not possible to construct shapes that reflect two different parameter sets within the same shape (a "bullet" shape, for instance, would need the squareness parameters on only one end of the superquadric).

To address this deficiency, the "blending" of two shapes has been defined. When two shapes are blended, the result is a shape that is a combination of the two original shapes. This blending is accomplished by the addition of a few parameters that describe how the shapes are combined.

A bullet shape could be defined by a blend of a sphere and a cylinder. Blends between shapes such as superquadrics and supertoroids are also possible, even though they differ topologically. The result would be a shape that has a hole which can appear depending on the blending parameters.

The vision applications for this include the ability to fit shapes with holes from range data. The initial shape would only develop a hole if the data reflects it. The graphics applications include a wider range of shapes to use for modeling systems (shapes with holes, for instance). This method for making blended shapes does not require the computation of intersection points between shapes, or the making of blending patches to connect the shapes.

23 Computation of Eye Movement of Two Agents in a Dialog Situation: Catherine Pelachaud

With Welton Becket and Chin Seah we have been working on eye movement. We are using Welton's finite automaton program: he has established a language which allows us to define easily actions and conditions of the automaton. The different states of the automaton are defined in a hierarchical way. Eye movement can be classified into 4 main subclasses depending on their role in the conversation.

1. Feedback: eye movement is used to collect and seek feedback.
2. Planning: corresponds to first phase of a turn when the speaker organizes his or her thoughts.
3. Comment: accompanies and comments speech. It occurs in parallel with accent, emphasis...
4. Control: controls the communication channel and it is used as a synchronization signal: responses may be demanded or suppressed by looking at the listener.

To each of these classes corresponds a "sub-finite-automaton." An automaton is defined by a set of nodes joined by arcs. A node is reached when a certain condition is true. When a condition is true, an action is performed. The top level of the automaton is:

```
NO(C1 or C2 or C3, spawn(FA-feedback), N1)
NO(C4 or C11, spawn(FA-planning), N2)
NO(C8 or C9 or C10, spawn(FA-comment), N3)
NO(C5 or C7, spawn(FA-control), N4)
```

where "C" corresponds to a condition, "spawn" activates a sub-finite-automaton, and "N" corresponds to the top node of the sub-automaton. Examples of conditions are:

- C1: existence of a grammatical pause
- C2: existence of a hesitation pause
- C3: speaker emitted a "within-turn-signal"
- C4: beginning of turn
- C5: end of turn
- C6: listener emitted a "back-channel signal"
- C7: listener emitted a "turn-requesting-signal"
- C8: existence of an accent
- C9: is the phrase a question
- C10: is the phrase an answer

- C11: is the turn of short duration
- ...

Example of actions (noted "A"; agent1, agent2 denote either the speaker or the listener) are:

- A1: agent1 gazes toward agent2
- A2: agent1 gazes away from agent2
- A3: agent1 gazes more at agent2
- A4: agent1 gazes less at agent2
- A5: break of the mutual gaze
- A6: establish mutual gaze

In parallel I have been working with Prof. Joseph Cappella. We first looked at data files provided by Prof. Cappella. These files are transcripts of a dialog situation between two agents. Each tenth of a second, twenty parameters are defined. They are:

1. number of turn (even turn numbers correspond to agent1, odd turn numbers correspond to agent2)
2. turn state (see definition below)
3. vocalization by agent1
4. vocalization by agent2
5. agent1 gazes at agent2's face
6. agent2 gazes at agent1's face
7. agent1 does adaptator's movement (hand on body, clothes...)
8. agent2 does adaptator's movement (hand on body, clothes...)
9. agent1 does speech related hand and arm gestures away from body. They correspond to illustrators.
10. agent2 does speech related hand and arm gestures away from body. They correspond to illustrators.
11. agent1 does body focus hand movements (such a rubbing hands together).
12. agent2 does body focus hand movements (such a rubbing hands together).
13. agent1 does a back-channel
14. agent2 does a back-channel

15. agent1 does a head-nod
16. agent2 does a head-nod
17. posture of agent1: (0: straight, 1: toward agent2, 2: away from agent2)
18. posture of agent2: (0: straight, 1: toward agent1, 2: away from agent1)
19. agent1 smiles
20. agent2 smiles

Turn-state is defined as follow:

- when agent1 has the floor:
  turn-state=0: agent1 and agent2 don't speak
  turn-state=1: agent1 speaks, agent2 doesn't speak
  turn-state=2: agent1 and agent2 speak
- when agent2 has the floor:
  turn-state=3: agent2 and agent1 don't speak
  turn-state=4: agent2 speaks, agent1 doesn't speak
  turn-state=5: agent2 and agent1 speak

An example of rules we looked at is:

- listener looks more at the speaker
- speaker and listener don't start and end smiling at the same time
- speaker smiles less than listener
- speaker smiles more when pausing than when talking
- speaker looks more when pausing than when talking
- if the turn is of short duration then there is more mutual gaze
- if there is mutual smile than there is less mutual gaze
- if speaker looks at the listener or speaker pauses, then listener does more smiles and head nods
- at the end of utterance there is more mutual gaze
- speaker looks less at the beginning of a turn

We are trying now to adapt these results to the finite-automaton definition. There are 24 possible states. Indeed, agent1 can gaze or not gaze. Agent1 can gaze while having the floor and talking or pausing... 1 denotes gazing at the other agent, 0 denotes no gaze; with the notation introduced earlier about turn-state, we have the following tables:

Gaze	
agent1	agent2
0	0
1	0
0	1
1	1

Floor		
agent1	agent2	floor
0	0	agent1
1	0	agent1
1	1	agent1
0	0	agent2
0	1	agent2
1	1	agent2

To each gaze-tuple corresponds a floor state: this gives us a 24-state machine. We have started looking at the data to find the probability of going from one state to another. This will be very useful for the definition of the finite-automaton defining to the gaze-behavior. These probabilities will correspond to weights on each node of the automaton. In parallel we have to define each function of eye movements by one of these 24 states.

References

[1] B. K. Alpert, "Sparse representation of smooth linear operators," Ph.D. dissertation, Department of Computer Science, Yale University, August 1990.

[2] B. K. Alpert, "A class of bases in L^2 for the sparse representation of integral operators," *SIAM Journal on Mathematical Analysis*, 24(1):246-262, January 1993.

[3] B. Alpert, G. Beylkin, R. Coifman, and V. Rokhlin, "Wavelet-like bases for the fast solution of second-kind integral equations," *SIAM Journal on Scientific Computing*, 14(1):159-184, January 1993.

[4] K. E. Atkinson, *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*, SIAM, Philadelphia, 1976.

[5] C. T. H. Baker, *The Numerical Treatment of Integral Equations*, Clarendon Press, Oxford, 1977.

[6] W. L. Briggs and V. E. Henson, "Wavelets and multigrid", *SIAM Journal on Scientific Computing*, 14(2):506-510, March 1993.

[7] I. Daubechies, *Ten Lectures on Wavelets*, SIAM, Philadelphia, 1992.

[8] S. J. Gortler, P. Schroder, M. F. Cohen, and P. Hanrahan, "Wavelet radiosity," *ACM Computer Graphics*, 221-230, August 1993.

[9] P. S. Heckbert, "Simulating global illumination using adaptive meshing," Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, June 1991.

[10] P. W. Hemker and H. Schippers, "Multiple grid methods for the solution of Fredholm equations of the second kind," *Mathematics of Computation*, 36(153):215-232, January 1981.

[11] W. Hackbusch, *Multi-Grid Methods and Applications*, Springer-Verlag, New York, 1985.

[12] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," *ACM Computer Graphics*, 20(4):133-142, August 1986.

[13] P. Schroder, S. J. Gortler, M. F. Cohen, and P. Hanrahan, "Wavelet projections for radiosity," *Proc. Fourth Eurographics Workshop on Rendering*, 105-114, June 1993.

[14] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg, "A global illumination solution for general reflectance distributions," *ACM Computer Graphics*, 25(4):187-196, July 1991.

A PaT-Nets: Welton Becket

Below is the initial documentation for PaT-Nets. There are many more features and access functions not described below — complete documentation will be available soon...

```
1) CREATING FSMS

   Create an FSM with DEFNET:

        (DEFNET fsm-name &optional :locals '(local1 local2...)
                                   :class-vars '(cvar1 cvar2...)
                                   :parents (list parent1 parent2)

            spec1
            spec2
              .
              .
              .)

    DEFNET defines an FSM class. Each instance of the class will
    have local1...localn as local variables available to all methods
    (all actions and conditions). Class-vars are shared among all
    instances of fsm-name. Parents should be the names of other
    FSM classes. The FSM class will inherit all methods (actions and
    conditions) and all nodes of the parents. Note that every
    FSM gets the FSM class as a parent automatically -- don't put
    it in the parent list.

   Specs can be:

      A) (DEFINIT (arg1 arg2 ... argn)
            code)

         Define initializer function to be called when a new FSM
         is instantiated from the FSM class. It takes the given
         arguments, which can be accessed DIRECTLY, as opposed
         to needing the VAL function like with instance and class
         vars (see below).

      B) (DEFACTION :action-name
            code)

         Create an action with the given name (which must begin with a ':').
         Code can be any arbitrary lisp code. The return value from
         an action is ignored.
```

C) (DEFCOND :condition-name
code)

Create a condition with the given name (which must begin with a ':'). A condition should return true if the arc containing it is to be taken and nil if it is not to be taken.

D) (DEFNODE node-name :action-name
(:cond1 state1) (:cond2 state2) ... (:condn staten))

Create a node with the given name (which doesn't have to start with a colon, though it can). When the node is entered it executes the :action-name action. Then it blocks on any waits executed in the action (see below). Then it looks for an arc with a true condition to follow, doing nothing if no arc is true (but not executing the action again -- there's an implicit wait-for-arc). Note that the first arc with a true condition is taken, so order is important (unless conditions are mutually exclusive). Each (:cond state) is an arc -- :cond is the name of a condition to evaluate, state is the name of a node to go to.

Special actions, conditions, and nodes:

:no-op = No action. This can go wherever an :action can go.

:default = Default condition. This ALWAYS evaluates to true -- any arcs appearing after an arc with the :default condition will never be considered.

exit = Exit state. This is how to exit an FSM, use exit as the name of the state to go to and the fsm will be exited on the current cycle.

E) (DEF-PROBNODE node-name :action
(prob1 state1) (prob2 state2) ... (probn staten))

Take each of the states with the given probabilities. Each prob-i should be a number in [0,1]. The prob-i should sum to less that or equal to 1.0. If no arc is taken (which happens only if prob-i sum to less that 1.0), then there's an implicit wait.

An example:

```
        (DEF-PROBNODE state1 :action1
            (0.1 state1) (0.7 state2) (0.2 state3))
```

which has a 10% chance of going to state1, a 70% chance of going to state2, and a 20% chance of going to state 3.

F) (DEF-WEIGHTNODE node-name :action
(weight1 state1) (weight2 state2) ... (weightn staten))

Similar to a DEF-PROBNODE, except the weight-i are scaled to sum to 1.0.

G) (DEF-RANDNODE node-name :action
state1 state2 state3 ... staten)

Takes one of the states randomly. Translates to a DEF-PROBNODE with each probability as 1/number-of-states.

H) (DEFMONITOR monitor-name :condition :action-name)

A monitor is evaluated on every cycle by an FSM before the state is advanced -- monitors are even evaluated when the FSM is in a wait. Monitor-name is the name of the monitor (which needn't start with a colon). :Condition is a condition which should evaluate to true when the monitor is to be executed. :Action-name is the name of an action that is executed when the monitor is true.

An Example:

```
(DEFNET samp-fsm  :locals '(a b)

   (DEFINIT (a b)
      (val a a)
      (val b b))

   (DEFACTION :inca
      (val a (1+ (val a))))

   (DEFACTION :incb
      (val b (1+ (val b))))

   (DEFCOND :c1 (< (val a) 5))

   (DEFCOND :c5 (< (val b) 5))

   (DEFNODE state1 :inca (:default state2))
```

```
        (DEFNODE state2 :no-op (:c1 state1) (:default state3))

        (DEFNODE state3 :incb (:default state4))

        (DEFNODE state4 :no-op (:c5 state3) (:default exit)))
```

---

2) CREATING FSM INSTANCES

Create an instance by sending and FSM class the message :new with any arguments required by the FSM initializer function. Like:

```
        (send fsm-class :new arg1 arg2 ... argn)
```

which returns the new FSM object. The object is automatically inserted in the active list of FSMs, and it will begin executing on the next cycle. Using the example above, do something like:

```
        (setf fred (send samp-fsm :new 1 2))
        (setf sam  (send samp-fsm :new 3 3))
```

To create a new instances of samp-fsm with different arguments bound to fred and sam. The two FSMs will begin executing in parallel in the next cycle.

---

3) ACTIONS AND CONDITIONS

The actions and conditions are actually methods in the FSM class in which they are defined (this is the reason they need to begin with a colon -- it's required by the class mechanism...).

A) LOCAL VARIABLES

Actions and conditions can access an FSMs local variables or class variables using VAR:

```
        (VAR a)  -- returns the value of FSM variable a

        (VAR a newval) -- sets the value of FSM variable to newval.
```

Note that a is not evaluated, but newval, if present, is evaluated.

B) SPAWNING NEW FSMS

An FSM can spawn a new fsm in an action just by using the standard FSM instantiation mechanism described above. An

example action that does this (and uses a wait, described below):

```
(DEFACTION :action1
   (val waiting-for
        (send another-fsm :new 5 6))
   (send self :wait-fsm (val waiting-for)))
```

C) WAITS

An action can post a set of waits that the fsm will wait for on exiting the action. The FSM will wait for the conjunction of all posted waits before continuing. Note that the FSM does not block until it exits the action. Each wait is signaled by the FSM sending a message to itself:

1) (SEND self :wait-fsm fsm-instance &key state)

Wait for an fsm to exit before continuing. If state is present it waits for FSM to pass through the given state (a node name) before continuing. Note that fsm-instance must be an instance of an FSM class and not the class itself. An example action:

```
(DEFACTION :action1
    (send self :wait-fsm (send myfsm :new 3 4)))
```

and to wait for a state:

```
(DEFACTION :action1
    (send self :wait-fsm (send myfsm :new 3 4) :state 'state2))
```

2) (SEND self :wait-time time)

Wait for a specific time in the motion system (in seconds). To wait 3 seconds before proceeding,

```
(send self :wait-time (+ (motion-time) 3.0))
```

3) (SEND self :wait-condition condition-fn)

Wait for an arbitrary lisp expression to evaluate to true. The condition-fn should be a function closure taking no arguments, like:

```
(send self :wait-condition #'(lambda () (< x 3.0)))
```

4) (SEND self :wait-motion motion-ptr)

Wait for a motion in the motion system to finish before proceeding. Motion-ptr must be a pointer to a motion, which can be found by name with the MOTION-FIND function. Example:

```
(send self :wait-motion (motion-find "default.ccube"))
```

5) (SEND self :wait-semaphore (s &key (priority 1.0)))

Wait on a semaphore s. Semaphores are instances of the semaphore class created like:

```
(setf s1 (send semaphore :new))
```

The FSM is placed on the semaphore's wait queue with the given priority (defaulting to 1 -- higher numbers for higher priority).

A semaphore is signaled (released) with:

```
(send self :signal-semaphore s)
```

Note that since waits don't block until after the action exits, no code dependent on having the semaphore should appear after the wait.

An example fsm using semaphores which waits on two semaphores then releases them in the next state:

```
(setf s1 (send semaphore :new))
(setf s2 (send semaphore :new))

(DEFNET semwait1

   (DEFACTION :action1
      (format t "in state 1~%")
      (send self :wait-semaphore s1)
      (send self :wait-semaphore s2))

   (DEFACTION :action2
      (format t "in state 2~%")
      (send self :signal-semaphore s1)
      (send self :signal-semaphore s2))

   (DEFNODE state1 :action1 (:default state2))
   (DEFNODE state2 :action2 (:default exit)))
```

D) OTHER MESSAGES AVAILABLE IN ACTIONS AND CONDITIONS

1) (send fsm-instance :exit &optional (exitcode 'finished))

Force an fsm instance (which can be self) to exit with a given exit code, one of 'finished or 'failed. The fsm kills any dependent processes and kills any fsms waiting on a particular state in itself.

2) (send fsm-instance :status)

Get the status of the fsm, one of:

```
'running  -- in active list
'idle     -- not running yet
'failed   -- exited with failed status
'finished -- exited with successful status
'waiting  -- waiting on conditions
```

3) (send fsm-instance :add-dependent dep-fsm-instance)

Tell the fsm-instance that dep-fsm-instance is a dependent process, so that when fsm-instance exits, it kills dep-fsm-instance.

B CVToJack Geometry Translation Program

B.1 Introduction

CVToJack is a program that translates geometry from a CADDS 4X version 6.2 system to the format used for *Jack*. In order to convert geometry from CADDS 4X to *Jack* the following is done:

1. The user generates ASCII geometry files of parts on the Computervision system using commands in the CADDS package.
2. Those files and procedure files are transferred to the computer system where *Jack* resides.
3. The geometry translator is then invoked to create a *Jack* psurf file or a *Jack* environment file.
4. When the entire process is complete, there will be a directory containing psurf files and another directory containing environment files.

B.2 Computervision Basics

The CADDS 4X tool generates parts which are a collection of entities. Each entity may be 3D geometry, 2D figures, display information, or instance information. Nodal figures (nfigs) are CADDS entities that are used for instancing other parts in a current part. Subfigures (Sfigs) are also used to instance other parts in a current part. For example, to create a room with a table and chairs: A set of walls and a floor and ceiling are created and saved as an nfig part, a table is created and saved as an nfig part, and a chair is created and saved as an nfig part. A new part is opened and the CADDS command insert nfig is used to place the walls, floor and ceiling part. The table is placed the same way. The chair is inserted several times to put several chairs about the table. From this example the top level part would be converted to a *Jack* environment file and each of the nfig parts would become psurf files.

Nfigs may also be used to place procedure file shapes in a part. A procedure file is a CADDS file that contains text with shapes and values for parameters of the shape in the file. Scale and variable information may be set in the nfig properties. Some construction tools in CADDS may automatically insert procedure file information during certain operations. All of this information is converted and becomes part of the psurf *Jack* file.

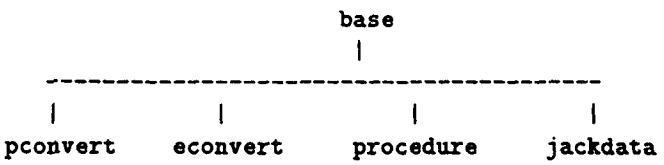
B.3 What is needed to convert?

The first item needed is for the CVToJack tools to be loaded on the machine on which you want to perform the conversion. It will run on an SGI workstation that also runs *Jack*.

A top level part in the CADDS data base that contains only nfig or sfig instances is needed to generate a *Jack* environment file. Procedure files may be instanced in an nfig of a top level part. If the procedure file has Nodal Line following parts, then Nodal Lines in the part are used to convert

the procedure file. If a top level part does not exist then the part that needs conversion may be saved as an nfig. A new part could then be generated that inserts this one nfig. In this case, the environment file would have only one psurf instanced and the psurf would be as large as the part. When possible the top level part should be made up of many nfig or sfig instances so that each of the psurf files will be smaller. When the top level file has entities other than nfigs, sfigs, or Nodal Lines, they will be ignored.

Also needed is a directory structure for copied files and for files generated by the conversion process. Whatever directory the conversion tools are run from is considered the base directory. Several required directories are needed at this level. These directories are **pconvert**, **econvert**, **procedure**, and **jackdata**. The **pconvert** directory is used to hold files that will be converted to psurfs. The **econvert** directory is used to hold files that will be converted to environment files. The **procedure** directory holds procedure text files. The **jackdata** directory stores environment files and psurf files after they have been converted. The following shows the directory structure:



B.4 Generating a *Jack* Environment File

A *Jack* environment file consists of orientation information and instances of psurf file data. A CV part file that is made of nfig instances may be converted to an environment file. When using the environment file option only nfig entities, sfig entities, and associated nlines of the CV part are looked at all others are ignored. The conversion process is as follows:

1. Using the CADDS 4X command, **OPEN PART** *name*, open the part.
2. Use the CADDS 4X command, **DO HARDFILE** *filename*, to store all screen information to a file.
3. Issue the CADDS 4X command, **LIST PART STATUS**, to dump to the screen (and the hardfile) part information including the units used. This command is optional and when not issued the units are assumed to be feet.
4. Issue the CADDS 4X command, **DUMP ENTITY**, to dump to the screen (and the hardfile) all the entity information for the part. This command must always be issued.
5. Use the CADDS 4X command, **DO HARDFILE**, to close the file. When closing the hardfile a filename is not needed on the command line. At this point a file named filename (or other user defined name) will reside in the **_bcd** directory that is under the CV create directory for the user.
6. Rename the file to correspond to the CV name such as
   mv dump1 a.b.c.part1.

It is necessary to do this in 2 steps since CV will convert "." in a filename to a directory level. You would not get **a.b.c.part1** in the **_bcd** directory. You would get a directory hierarchy with directory **a** in **_bcd**, directory **b** in **a**, directory **c** in **b**, and the file **part1** in directory **c**.

7. Copy the file over to the **econvert** directory on the conversion machine.
8. Run the conversion tool on the file with the environment file option.
9. You will be prompted for a filename to store nfig names not found in the **jackdata** directory. This list tells the user which parts need to be generated into psurf files. All previously generated psurf files do not have to be redone.
10. On completion a file named *filename*.**env** will be generated in the **jackdata** directory.

B.5 Generating a *Jack* Psurf File

A *Jack* psurf file consists of geometry information in a local coordinate system. A CV part file that is made of entities and procedure file data may be converted to a psurf file.

The conversion process is as follows:

1. Using the CADDS 4X command, **OPEN PART** *name*, open the part.
2. Use the CADDS 4X command, **DO HARDFILE** *filename*, to store all screen information to a file.
3. Issue the CADDS 4X command, **LIST PART STATUS**, to dump to the screen (and the hardfile) part information including the units used. This command is optional and when not issued the units are assumed to be inches.
4. Issue the CADDS 4X command, **DUMP ENTITY**, to dump to the screen (and the hardfile) all the entity information for the part. This command must always be issued.
5. Use the CADDS 4X command, **DO HARDFILE**, to close the file. When closing the hardfile a filename is not needed on the command line. At this point a file named *filename* will reside in the **_bcd** directory that is under the CV create directory for the user.
6. Rename the file to correspond to the CV name such as

   mv dump1 a.b.c.part1.

   It is necessary to do this in 2 steps since CV will convert "." in a filename to a directory level. You would not get **a.b.c.part1** in the **_bcd** directory. You would get a directory hierarchy with directory **a** in **_bcd**, directory **b** in **a**, directory **c** in **b**, and the file part1 in directory **c**.
7. Copy the file over to the **pconvert** directory on the conversion machine.
8. Run the conversion tool on the file with the psurf file option.
9. You will be prompted for a filename to store nfig names and procedure file names not found in the **pconvert** or **procedure** directories. This list tells the user which parts need to be copied to the **procedure** directory and the **pconvert** directory.

10. On completion a file named *filename*.**pss** will be generated in the jackdata directory when all the procedure files instanced are found and all the instanced nfig parts are found. If for some reason the user does not want to copy these files over, he or she may create empty files in the appropriate directories. The converter will find the filenames and create a psurf file.

C Hierarchical Shape Representation Using Locally Adaptive Finite Elements: A Model-Based Approach: Eunyoung Koh, Dimitri Metaxas and Norman Badler

Hierarchical Shape Representation Using Locally Adaptive Finite Elements: A Model-Based Approach

Abstract

This paper presents a physics-based algorithm for hierarchical shape representation using deformable models with locally adaptive finite elements. We develop our technique using dynamic deformable models which support local and global deformations. The dynamic deformable models express global deformations with a few parameters which represent the gross shape of an object, while local deformations capture shape details of objects through their many local parameters. Our new adaptive finite element algorithm ensures that during subdivision the desirable finite element mesh generation properties of conformity, non-degeneracy and smoothness are maintained. Through our algorithm, we locally subdivide the triangular finite elements based on the distance between the given datapoints and the model. In this way, we can very efficiently and accurately represent the shape of an object with a resulting small number of model nodes. Furthermore, using our locally adaptive subdivision algorithm in conjunction with our model's global deformations we construct a hierarchical representation of the shape of 3D data with multiple levels of surface detail.

Keywords: Adaptive Finite Elements, Hierarchical Representation, Physics-Based modeling, Deformable Models, Model-Based Approach.

1 Introduction

Recovering the shapes and motions of 3D objects from visual data is a primary goal of low and intermediate level vision. While simple shape primitives such as spheres, cylinders and polyhedra have the advantage of representing shapes with only a few parameters, their representational power is inadequate when it comes to natural objects. The shapes of most natural objects are complex and irregular and can't be represented in terms of simple shape primitives. Spline models that deform locally subject to continuity constraints are well suited to free-form shape representation by virtue of their relatively distributed parameters sets which provide many local degrees of freedom.

Recently [15, 7] we have created a new family of modeling primitives by developing a mathematical approach that allows the combination of global and local deformations. Our primitives include global deformation parameters which represent the salient shape features of natural parts and local deformation parameters which capture shape details. More specifically we developed hybrid models whose underlying geometric structure allows the combination of parametric models (superquadrics, spheres, cylinders), parameterized global deformations (bends, tapers, twists, shears, etc.) and local spline free-form deformations. In this way the descriptive power of our models is a superset of the descriptive power of locally deformable models [17, 16] and globally deformable models [10].

Through the application of Lagrangian mechanics, we develop a method [8] to systematically convert the geometric parameters of the solid primitive, the global (parameterized) and local (free-form) deformation parameters, and the six degrees of freedom of rigid-body motion into generalized coordinates or dynamic degrees of freedom. More precisely, our method applies generally across all well-posed geometric primitives and deformations, so long as their equations are differentiable. The distinguishing feature of our approach is that it combines the parameterized and free-form modeling paradigms within a single physical model. Thus our models based on forces that the datapoints exert on the model and the prescribed mass distributions, elasticities and energy dissipation rates change positions and orientations and deform away from their rest positions so as to conform to the data set.

Even though the formulation of global deformations is independent of the number of model nodes used, local deformations require a tessellation of the model surface into a grid of finite elements. The quality of the model fit to the data depends on the number of the finite elements used. In our framework we assumed that the size of the finite element grid remained constant throughout the model fit to the data. Clearly this is a serious limitation in case of model fitting applications where the user assumes no prior knowledge of the complexity of the given data.

Almost all of the current physics-based shape recovery algorithms which use 3D or surface models assume fixed-size grids [17, 10, 8, 19, 3, 4]. Recently [18] proposed a technique for adaptive subdivision of meshes consisting of nodal masses interconnected by adjustable springs. Even though their technique works well for adaptive fixed-size meshes, in case of local subdivision of the mesh a computationally expensive constraint procedure has to be applied to ensure that the triangular structure of the mesh is maintained. In [5] an adaptive subdivision algorithm is presented for nonrigid motion analysis which uses planar triangular patches. Planar patches though are not sufficient for representing highly convoluted surfaces and ensuring desirable continuity properties (in the finite element technique the continuity level of the solution is determined by the energy expression associated with each finite element and the use

of appropriate elements). Finally in [14] an adaptive mesh generation algorithm is presented for surface reconstruction. Even though the algorithm supports local subdivision, it does not use any underlying 3D model, can't guarantee the level of smoothness of the solution since it is based on geometry and special algorithms need to be employed to deal with cracks often occurring during subdivision.

In this paper we extend our physics-based framework for shape estimation by developing a new technique which allows the robust local adaptive subdivision of an initial finite element grid based on the distances of the given datapoints to the model surface. Our technique is an adaptation of Rivara's local refinement process for finite element grids [11]. Starting from a small number of finite elements tessellating the model surface, the model locally subdivides based on the above criterion so as to improve the model fitting. In this way with a resulting small number of model nodes we can very efficiently and accurately represent the shape of an object. In order to further improve the fit of the model to datapoints we also modify our previously developed force assignment algorithm [15] so that each datapoint gets assigned to a point within a finite element whose distance from the datapoint is minimum, instead of to a model node. Once the force is assigned it is appropriately distributed to the nodes of the corresponding finite element according to the finite element theory.

Our local subdivision algorithm utilizes the properties of triangles bisected by the longest side. That is, the interior angles of the refined triangles do not go to zero as the level of subdivision goes to infinity. Also this triangulation improves the shape regularity of the subdivided triangles as the subdivision proceeds. The local subdivision algorithm can be shown to satisfy conformity, non-degeneracy and smoothness which are desirable properties for finite element meshes and ensure the accuracy of the solution.

Our locally adaptive subdivision algorithm combined with the global deformations of our dynamic models, allows the reconstruction and representation of shape hierarchically. Our shape hierarchy consists of using global deformations only, then then global and one course level of local deformation and finally global and local deformations with various levels of local deformations extracted from our locally adaptive subdivision algorithm. The hierarchical representation with gradual object surface detail also has important applications in computer graphics. There, it can be used to resolve the issue of handling excessive detail when the object is far from the viewer or it is moving very fast.

Using our new local subdivision algorithm we present shape recovery experiments with real and synthetic range data sets which run at interactive rates on an Iris Crimson workstation with VGX graphics. In this way we not only represent accurately the shapes of objects, but also efficiently since a lot fewer model nodes than the given datapoints are required.

2 Overview

Section 3 reviews the formulation of deformable models. Section 4 presents our local adaptive subdivision algorithm which also includes the description of the finite elements used and our new force assignment technique. Section 5 presents experimental results that demonstrate the recovery of object shape from 3D range data using our deformable models. Section 6 draws conclusions from our work.

3 Deformable Models

In this section we briefly review the general formulation of deformable models; further detail can be found in [15, 8].

3.1 Geometry: Global and Local Deformations

Geometrically, the models developed in this paper are closed surfaces in space whose intrinsic (material) coordinates are $\mathrm{u} = (u, v)$, defined on a domain Ω. The positions of points on the model relative to an inertial frame of reference Φ in space are given by a vector-valued, time varying function of u: $\mathbf{x}(\mathrm{u}, t) = (x_1(\mathrm{u}, t), x_2(\mathrm{u}, t), x_3(\mathrm{u}, t))^T$, where ^{T} is the transpose operator. We set up a noninertial, model-centered reference frame ϕ and express these positions as:

$$\mathbf{x} = \mathbf{c} + \mathbf{R}\mathbf{p}, \tag{1}$$

where $\mathbf{c}(t)$ is the origin of ϕ at the center of the model and the orientation of ϕ is given by the rotation matrix $\mathbf{R}(t)$. Thus, $\mathbf{p}(\mathrm{u}, t)$ denotes the canonical positions of points on the model relative to the model frame. We further express $\mathbf{p}$ as the sum of a reference shape $\mathbf{s}(\mathrm{u}, t)$ and a displacement function $\mathbf{d}(\mathrm{u}, t)$, i.e., $\mathbf{p} = \mathbf{s} + \mathbf{d}$.

The ensuing formulation can be carried out for any reference shape given as a parameterized function of u. Based on the shapes we want to recover, we first consider the case of superquadric ellipsoids [1], which are given by the following formula:

$$\mathbf{e} = (e_1, e_2, e_3)^T; e_1 = aa_1 C_u{}^{\epsilon_1} C_v{}^{\epsilon_2}, e_2 = aa_2 C_u{}^{\epsilon_1} S_v{}^{\epsilon_2}, e_3 = aa_3 S_u{}^{\epsilon_1}, \tag{2}$$

where $-\pi/2 \leq u \leq \pi/2$ and $-\pi \leq v < \pi$, and where $S_w{}^{\epsilon} = \mathrm{sgn}(\sin w)|\sin w|^{\epsilon}$, and $C_w{}^{\epsilon} = \mathrm{sgn}(\cos w)|\cos w|^{\epsilon}$, respectively. Here, $a \geq 0$ is a scale parameter, $0 \leq a_1, a_2, a_3 \leq 1$ are aspect ratio parameters, and $\epsilon_1, \epsilon_2 \geq 0$ are "squareness" parameters.

We then combine linear tapering along principal axes 1 and 2, and bending along principal axis 3 of the superquadric $\mathbf{e}$[1] into a single parameterized deformation $\mathbf{T}$, and express the reference shape as:

$$\mathbf{s} = \mathbf{T}(\mathbf{e}, t_1, t_2, b_1, b_2, b_3) = \begin{pmatrix} (\frac{t_1 e_3}{a a_3 w} + 1)\, e_1 + b_1\, cos(\frac{e_3 + b_2}{a a_3 w} \pi b_3) \\ (\frac{t_2 e_3}{a a_3 w} + 1)\, e_2 \\ e_3 \end{pmatrix}, \tag{3}$$

where $-1 \leq t_1, t_2 \leq 1$ are the tapering parameters in principal axes 1 and 2, respectively, and where b_1 defines the magnitude of the bending and can be positive or negative, $-1 \leq b_2 \leq 1$ defines the location on axis 3 where bending is applied and $0 < b_3 \leq 1$ defines the region of influence of bending. Our method for incorporating global deformations is not restricted to only tapering and bending deformations. Any other deformation that can be expressed as a continuous parameterized function can be incorporated as our global deformation in a similar way. We collect the parameters in $\mathbf{s}$ into the parameter vector

[1] These coincide with the model frame axes x, y and z respectively

$\mathbf{q}_s = (a, a_1, a_2, a_3, \epsilon_1, \epsilon_2, t_1, t_2, b_1, b_2, b_3)^T$.

Local, finite element basis functions are the natural choice for representing the local deformations [15, 8]. The elements have a node at each of their corners. The generalized coordinates of the finite element basis functions are the nodal variables—a vector $\mathbf{q}_{d_i}$ associated with each node i of the model. If we collect the generalized coordinates into a vector of degrees of freedom $\mathbf{q}_d = (\ldots, \mathbf{q}_{d_i}^T, \ldots)^T$, we can write $\mathbf{d} = \mathbf{S}\mathbf{q}_d$, where $\mathbf{S}$ is the shape matrix whose entries are the finite element basis functions. In a subsequent section we will give more details about the finite elements we use.

3.2 Kinematics and Dynamics

The velocity of points on the model is given by,

$$\dot{\mathbf{x}} = \dot{\mathbf{c}} + \dot{\mathbf{R}}\mathbf{p} + \mathbf{R}\dot{\mathbf{p}} = \dot{\mathbf{c}} + \mathbf{B}\dot{\boldsymbol{\theta}} + \mathbf{R}\dot{\mathbf{s}} + \mathbf{R}\mathbf{S}\dot{\mathbf{q}}_d, \tag{4}$$

where $\boldsymbol{\theta} = (\ldots, \theta_i, \ldots)^T$ is the vector of rotational coordinates of the model and $\mathbf{B} = [\ldots \partial(\mathbf{R}\mathbf{p})/\partial\theta_i \ldots]$. Furthermore, $\dot{\mathbf{s}} = [\partial\mathbf{s}/\partial\mathbf{q}_s]\dot{\mathbf{q}}_s = \mathbf{J}\dot{\mathbf{q}}_s$, where $\mathbf{J}$ is the Jacobian of the superquadric with global deformations function. We can therefore write

$$\dot{\mathbf{x}} = [\mathbf{I}\ \mathbf{B}\ \mathbf{R}\mathbf{J}\ \mathbf{R}\mathbf{S}]\dot{\mathbf{q}} = \mathbf{L}\dot{\mathbf{q}}, \tag{5}$$

where $\mathbf{q} = (\mathbf{q}_c^T, \mathbf{q}_\theta^T, \mathbf{q}_s^T, \mathbf{q}_d^T)^T$, with $\mathbf{q}_c = \mathbf{c}$ and $\mathbf{q}_\theta = \boldsymbol{\theta}$.

Our goal when fitting the model to visual data is to recover the vector of degrees of freedom $\mathbf{q}$. Our approach carries out the coordinate fitting procedure in a physically-based way—by enabling the data to apply traction forces to the surface of the model [15]. We can make our model dynamic in $\mathbf{q}$ by introducing mass, damping, and a deformation strain energy. In applications to vision [8], it makes sense to simplify the motion equations while preserving useful dynamics by setting the mass density $\mu(\mathrm{u})$ to zero to obtain

$$\mathbf{D}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \mathbf{f}_q, \tag{6}$$

where $\mathbf{D}$ and $\mathbf{K}$ are the damping and stiffness matrices respectively, and where $\mathbf{f}_q(\mathrm{u}, t)$ are the generalized external forces associated with the degrees of freedom of the model. The above equation yields a model that has no inertia and comes to rest as soon as all the applied forces equilibrate or vanish. We also decouple the equations by assuming that $\mathbf{D}$ is diagonal and constant over time. For fast interactive response, we employ a first-order Euler method to integrate (6). Note that we represent the rotation component $\mathbf{q}_\theta$ as a quaternion and that we never assemble a finite element stiffness matrix, but compute $\mathbf{K}\mathbf{q}$ in an element-by-element fashion for the local deformation degrees of freedom [8].

4 Locally Adaptive Finite Elements

In this section we will describe the local strain energy and the finite elements used, our algorithm for assigning forces from datapoints to points on the model, our criterion to locally select elements for subdivision and our local finite element subdivision algorithm.

4.1 Finite Elements with C^1 Continuity.

For the applications in this paper we select a strain energy and the appropriate finite elements that guarantee C^1 continuity. In particular we present new efficient shape functions. A thin plate under tension deformation energy, suitable for C^1 continuous model surface, is given by the functional

$$\mathcal{E}_p(\mathbf{d}) = \int w_{20}\left(\frac{\partial^2 \mathbf{d}}{\partial u^2}\right)^2 + w_{11}\left(\frac{\partial^2 \mathbf{d}}{\partial u \partial v}\right)^2 + w_{02}\left(\frac{\partial^2 \mathbf{d}}{\partial v^2}\right)^2 + w_{10}\left(\frac{\partial \mathbf{d}}{\partial u}\right)^2 + \; w_{01}\left(\frac{\partial \mathbf{d}}{\partial v}\right)^2 + w_{00}\mathbf{d}^2 \, du. \tag{7}$$

The nonnegative weighting functions w_{ij} control the elasticity of the material. Increasing w_{01} and w_{10} makes the deformations have more membrane properties, while increasing the w_{20}, w_{11} and w_{02}, the deformations behave more like a thin plate. In our implementation, however, we reduce these functions to scalar stiffness parameters $w_{ij}(\mathrm{u}) = w_{ij}$.

To approximate the above strain energy (7) we use triangular finite elements [8] whose shape functions are tensor products of one-dimensional Hermite polynomials [20] which are given by the formulas

$$\begin{aligned} H_1^0(\xi) &= 1 - 3\xi^2 + 2\xi^3 \qquad H_1^1(\xi) = \xi(\xi-1)^2 \\ H_2^0(\xi) &= \xi^2(3-2\xi) \qquad H_2^1 = \xi^2(\xi-1) \end{aligned} \tag{8}$$

where the subscripts are related to the two endpoints of the one-dimensional segment and the superscripts, 0 and 1, denote the association of a basis function to a nodal variable or a nodal derivative respectively. These new finite elements are much more computationally efficient than the ones described in [8, 9].

Using the above finite elements and the corresponding shape functions we compute from the strain energy (7)) the stiffness matrix $\mathbf{K}_{dd}$ through a technique based on the theory of elasticity and demonstrated for the case of a loaded membrane deformation energy in [8].

4.2 Force Assignment

In our applications, for each given range datapoint $\mathbf{z}$ we want to find a point u_z on the model that minimizes the distance d between $\mathbf{z}$ and the model, where: $d(\mathrm{u}) = \|\mathbf{z} - \mathbf{x}(\mathrm{u})\|$. A brute-force approach to the above minimization problem that worked well in our previous efforts [15, 8] is to select from all the model nodes the one that minimizes d. The above approach is inadequate for our local finite element subdivision algorithm since it only takes into account model nodes in computing $d(\mathrm{u})$. In this paper we will use the following algorithm which is the most accurate possible, given that we approximate the model surface with finite elements and there is no analytic formula for $\mathbf{x}(\mathrm{u})$.

Algorithm

In this algorithm for each datapoint $\mathbf{z}$ we perform the following minimization. According to the finite element theory, we use the elemental shape functions N_i to approximate an element's surface and compute the point in each finite element whose distance minimizes

$$d(\mathrm{u}) = \min^j \|\mathbf{z} - \mathbf{x}(\mathrm{u})\| = \min^j \left\| \mathbf{z} - \sum_{i=1}^{n} N_i(\mathrm{u})\mathbf{x}_i \right\|, \tag{9}$$

where N_i is the shape function corresponding to node i whose nodal position is $\mathbf{x}_i$ and j is a finite element

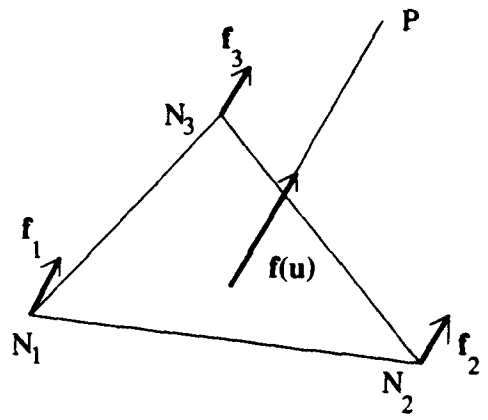


Figure 1: Extrapolation of force to the element nodes.

from the set E of finite elements that comprise the model surface. ¿From these computed model points we select the one u_z whose distance $d(\mathrm{u}_z)$ from the datapoint is the minimum of all the computed distances. The complexity of the algorithm is $O(mn)$, where m is the number of finite elements used and n is the number of given datapoints.

We then assign to u_z the following force

$$\mathbf{f}(\mathrm{u}_z) = \beta(\mathbf{z} - \mathbf{x}(\mathrm{u}_z)), \tag{10}$$

based on the separation between the datapoint $\mathbf{z}$ in space and the force's point of influence u_z on the model's surface and where β is the strength of the force given as an input parameter. We then extrapolate $\mathbf{f}(\mathrm{u}_z)$ to the element nodes using the formula

$$\mathbf{f}_i = N_i(u_z)\mathbf{f}(u_z), \tag{11}$$

where N_i is the shape function that corresponds to node i and $\mathbf{f}_i$ is the extrapolated value of $\mathbf{f}(u_z)$ to node i. (Fig. 1).

4.3 Criterion for Finite Element Subdivision

We use a criterion based on the distance from the datapoints to the finite elements and the curvature inherent in the datapoints, to decide whether an element or elements should be subdivided. We first compute using the above algorithm the point u_z on the model whose distance $d(\mathrm{u}_z)$ is the minimum from the given datapoint $\mathbf{z}$. If

$$d(\mathrm{u}_z) > \tau_d, \tag{12}$$

where τ_d is a threshold, we subdivide the elements that this nearest model point is on. We distinguish the following three cases.

1. If u_z lies inside an element, then the element is selected for subdivision.

2. If u_z lies on an edge, then the two adjacent elements to the edge are subdivided.

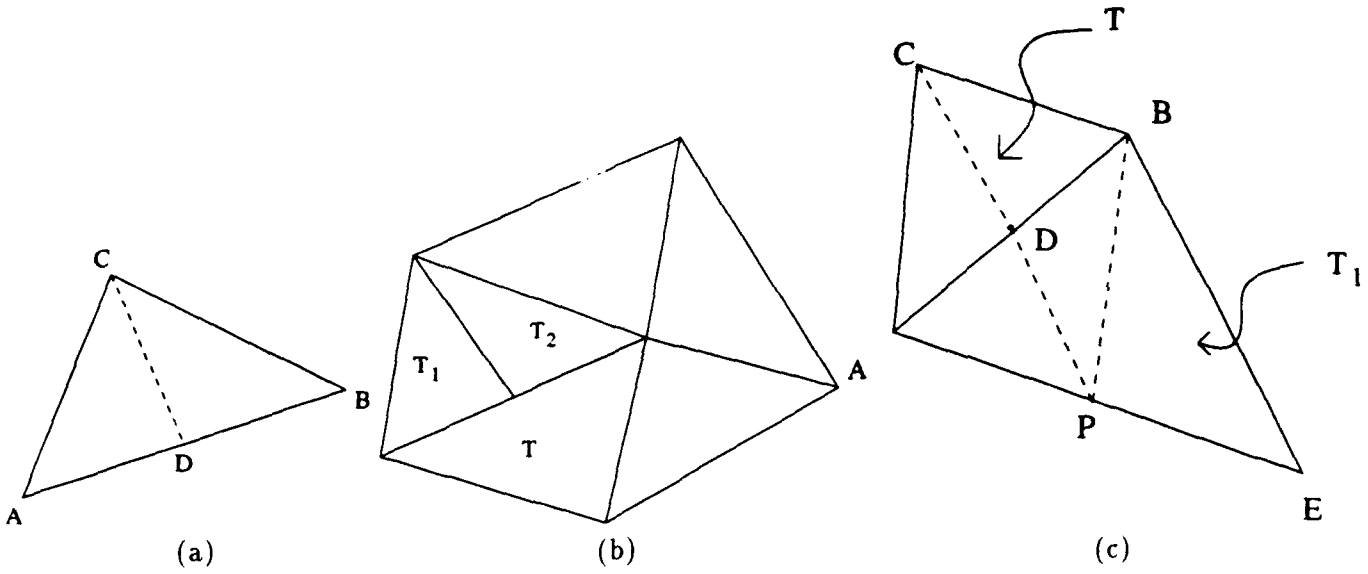


Figure 2: Various subdivision examples. (a) Subdivision of a triangle by the longest edge, (b) An example of non-conforming triangulation, (c) An illustration of conforming operation.

3. If u_z is a model node, then all the elements adjacent to the node are subdivided.

Once the above criterion is satisfied we subdivide the chosen elements and apply the following subdivision algorithm to ensure that the resulting grid has properties necessary for the application of the finite element method. It is worth mentioning that given that curvature calculation is very sensitive to noise and that we want to use our technique in case of sparse data, we did not consider using the data curvature as a criterion for subdivision. Also our criterion for subdivision does not use the model surface curvature, because the chosen finite elements ensure a smooth C^1 continuous solution.

4.4 Subdivision Algorithm

Our subdivision algorithm is essentially based on Rivara's local refinement algorithm [11]. Tne subdivision algorithm has the following two basic steps.

Step 1: Bisection Operation

In the first step the chosen finite element based on the above criterion performs a *bisection operation* as follows: let T be a triangle with vertices A, B, and C; a natural way of subdividing the triangle T into two triangles for finite element methods [12] is to bisect it along its longest edge; let AB be the longest edge of T, and D the midpoint of AB; then T is subdivided into two triangles, ADC and BCD as shown in Fig. 2(a).

This subdivision has been shown to provide properties desirable for use in finite element applications. First, none of the interior angles of the refined triangles will become obtuse as the level of subdivision increases. Rosenberg and Stenger [12] proved that if α_i is the smallest angle of the triangulation obtained by the i-th iterative subdivision, then $\alpha_i \geq \frac{\alpha_0}{2}$ for any i, where α_0 is the smallest interior angle of the initial triangulation. Second, the subdivision improves the shape regularity of the triangles, that is, the triangles become approximately equilateral as the level of subdivision increases [13].

Step 2: Conforming Operation

The second part of the algorithm ensures that the resulting finite element grid generates properties necessary for application of the finite element method. A triangulation is defined to be conforming if any two adjacent triangles must share either a common vertex or a common edge [20]. In Fig. 2(b), the triangulation is not conforming, because conformity is violated between T_1 and T, and between T_2 and T. In the finite element method, we must maintain the continuity across inter-element boundaries, i.e., it is necessary to maintain the conformity of the triangulation.

In Fig. 2(c), if we introduce a new node, D, as a result of bisecting element T, the element, T_1, adjacent to the subdivided edge AB becomes non-conforming. In order to ensure conformity, further subdivision must be performed on T_1 along the edge common with midpoint D. However, it is possible that the common edge may not be the longest edge of T_1. Therefore, this subdivision will cause the triangulation to lose the aforementioned properties of shape regularity.

To remedy this problem, we take the following approach in subdividing element T_1 as shown in Fig.2(c). We first bisect T_1 by its longest edge, AE, at its midpoint, P. If AE is the common edge, then we stop subdividing. Otherwise, we further subdivide T_1 by connecting P to the midpoint D of AB. As a result of this process, conformity is preserved and the subdivision will not produce triangles with obtuse angles. This process is called a *conforming operation.*

In our local subdivision algorithm, the conforming operation is performed whenever subdivision of an element causes non-conformity. The conforming operation, however, may create new non-conformity. In order to ensure the conformity this conforming operation is recursively applied until the triangulation becomes entirely conforming. This recursive process is guaranteed to stop because there is only a limited number of triangles to start with. Fig. 3 illustrates an example of applying a series of conforming operations which were necessary because of propagating non-conformity.

Based on the above description, our local subdivision algorithm can be given as follows:

```
While the subdivision set is not empty
BEGIN
    Set T as a triangle from the subdivision set.
    Remove T from the subdivision set.
    If T has not been bisected then
    BEGIN
        Set E_longest as the longest edge of T.
        Subdivide E_longest.
        Bisect T by E_longest.
        Set T' as the adjacent triangle of T by the edge E_longest.
        Conform(T', E_longest).
    END
END
```

We define the conforming operation Conform(T', E') as

Set $E'_{longest}$ as the longest edge of T'

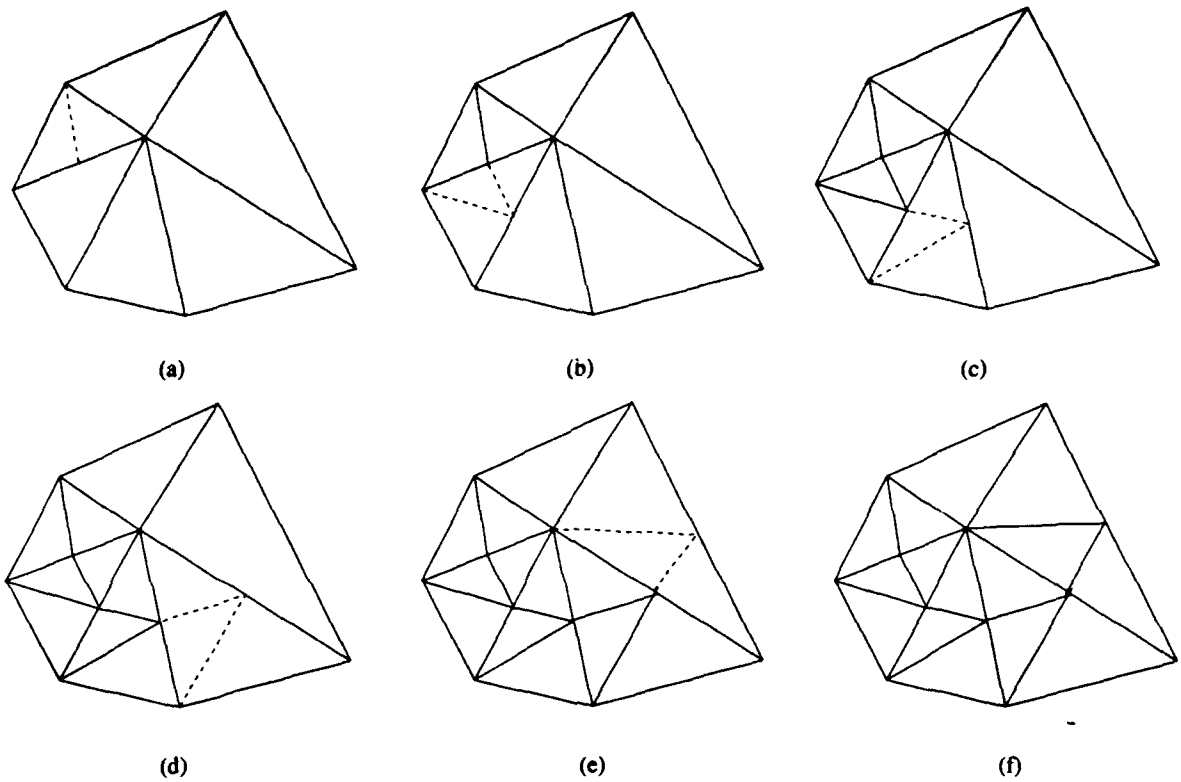


Figure 3: An example of recursive local subdivision.

if $E'_{longest}$ is the same as E' then
BEGIN
    Bisect T' by E'.
    Return.
END
Subdivide $E'_{longest}$.
Bisect T' by $E'_{longest}$.
Set $\hat{T}$ as one of the sub-triangles from the previous step which contains the edge E'.
Bisect $\hat{T}$ by E'.
Set $\tilde{T}$ as the adjacent triangle of T' by the edge $E'_{longest}$.
Conform($\tilde{T}$, $E_{longest}$).

In summary, our local subdivision algorithm satisfies several desirable properties for finite element mesh generation. They are: 1) conformity: any two adjacent elements share only either a node or an edge; 2) non-degeneracy: the triangulation maintains the shape regularity of the refined elements, i.e., no angles of the element become obtuse, and the triangles tend to be approximately equilateral; 3) smoothness: there is no abrupt size difference between adjacent elements, and hence the transition between small and large elements is smooth.

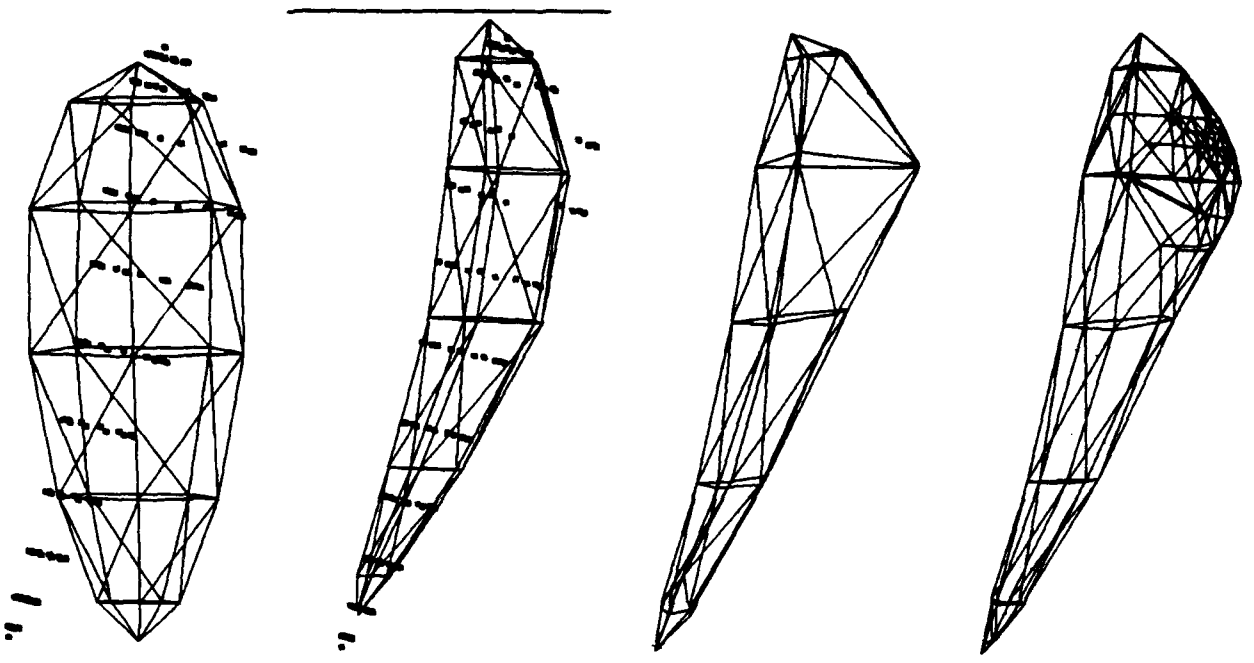


Figure 4: Fitting of model to squash-shaped data. (a) Model initialization, (b) Intermediate step of model fitting with apparent global deformations, (c) Model fitted to data without local subdivision, (d) Model fitted to data after three levels of local subdivision.

5 Experiments

We have carried out various experiments to test our locally adaptive finite element algorithm. These include 3D range data taken from the NRCC range image database, the Cyberware, Inc., 3D digitizer, a human head model provided by Viewpoint, Inc., and synthetic 3D data sampled from the surfaces of deformable superquadrics. Using our new force assignment algorithm the models deform globally and locally and subdivide locally to fit the given data. Our experiments run at interactive rates on an R4000 Iris Crimson workstation with VGX graphics. In all the experiments we used a time step size equal to 1.0×10^{-5} for the Euler method and a unit damping matrix $\mathbf{D}$, while the threshold for local subdivision was $\tau_d = 0.05$.

In Fig. 4 we fit a deformable model with 27 nodes to 123 3D datapoints sampled from the surface of a squash-like model undergoing deformations, where the local deformations simulated a loaded membrane. The local deformation stiffness parameters of the model were $w_{00} = 0.5$, $w_{01} = 0.5$, $w_{10} = 0.5$, $w_{02} = 0.0$, $w_{11} = 0.0$ and $w_{20} = 0.0$ (the latter three were set to zero to give membrane properties to the local deformations of our model). The initial model was an ellipsoid ($\mathbf{q}_s = (2.6, 0.35, 0.35, 0.9, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)^T$) and the force strength parameter was $\beta = 100.0$. Fig. 4(a) shows a view of the range data and the initial model, while Fig. 4(b) shows an intermediate step in the fitting of the model to the data where the global deformations are apparent. Fig. 4(c) shows the model fitted to the data without local subdivision, while Fig. 4(d) shows the final model fitted to the data after three levels of local subdivision. The improvement in the shape is obvious, as well as the new triangles formed as a result of the subdivision. The new final number of model nodes is 46, which is significantly smaller than the number of given datapoints.

To better demonstrate the creation of new triangles through our algorithm, Fig. 5 shows in parameter

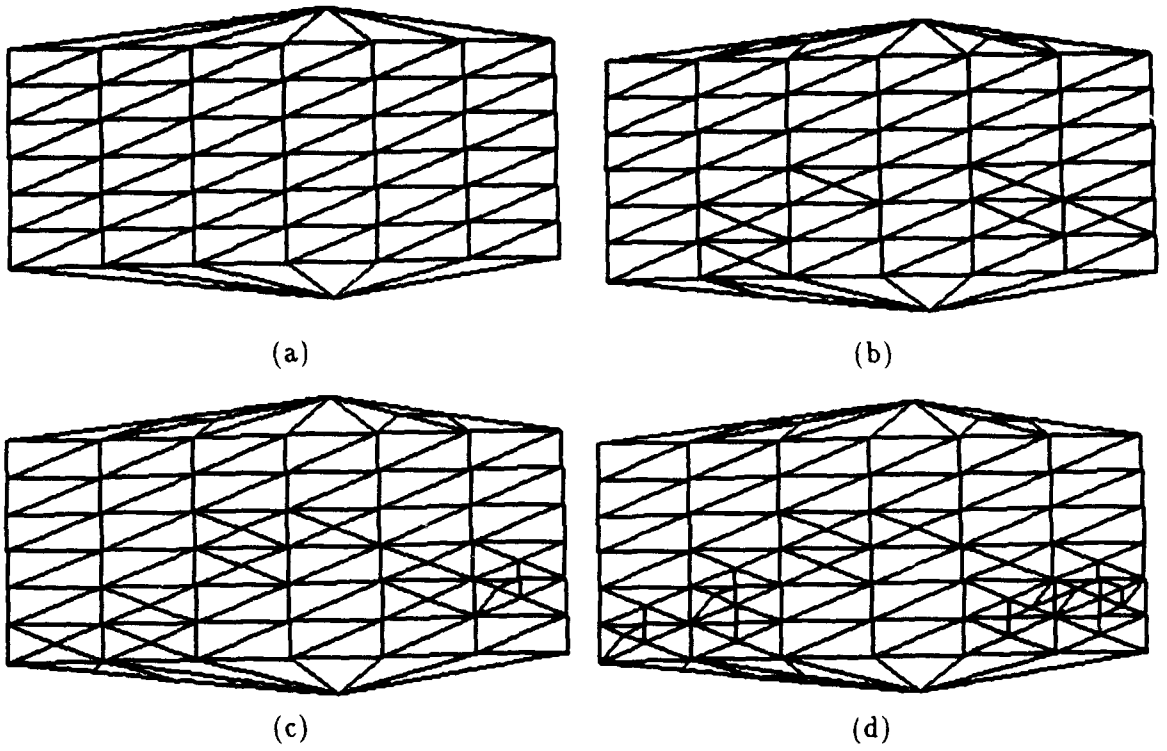


Figure 5: (a) Initial triangular mesh, (b), (c) and (d) Resulting triangular meshes at the end of each of the three subsequent local subdivision levels.

space the triangles formed after each level of subdivision in an experiment analogous to the above, where the initial number of model nodes was 51. Fig. 5(a) shows the initial triangular mesh, while Figs. 5(b), (c) and (d) show the resulting triangular meshes at the end of each of the three subsequent local subdivision levels.

In the following experiments, the model's fitting errors with respect to the input data are measured, where fitting errors are defined in terms of the distances from the original input data to the fitted deformable model. The fitted model is obtained by fitting the deformable model to input data until the movement of each element becomes relatively nonexistent. That is, we audit the difference between two positions of each element obtained from two consecutive iterations during fitting, and we allow the model to continue fitting until the maximum difference in the model elements is less than a given threshold value.

In the first experiment, a deformable model with 66 nodes was fitted to 857 3D data points sampled from a biomedical image of the left part of human lung (Fig. 6(a)). The initial fitting model was an ellipsoid (Fig. 6(b)). Fig. 6(c) shows the model after global deformations. Fig. 6(d) shows the model after local deformations. Fig. 6(e) shows the model with 243 nodes after three levels of our locally adaptive subdivision. Comparison of Fig. 6(d) and Fig. 6(e) shows that the subdivision was concentrated in the boundary area with more features.

Table 1 shows error statistics collected after fitting the deformable models to the input data of human lung. For one model we allowed locally adaptive subdivision while we superseded subdivision on other two models. We show the number of nodes in the model, the number of polygons in the model, the number of iterations, the mean error value, the maximum error value, the standard deviation and the variance of the

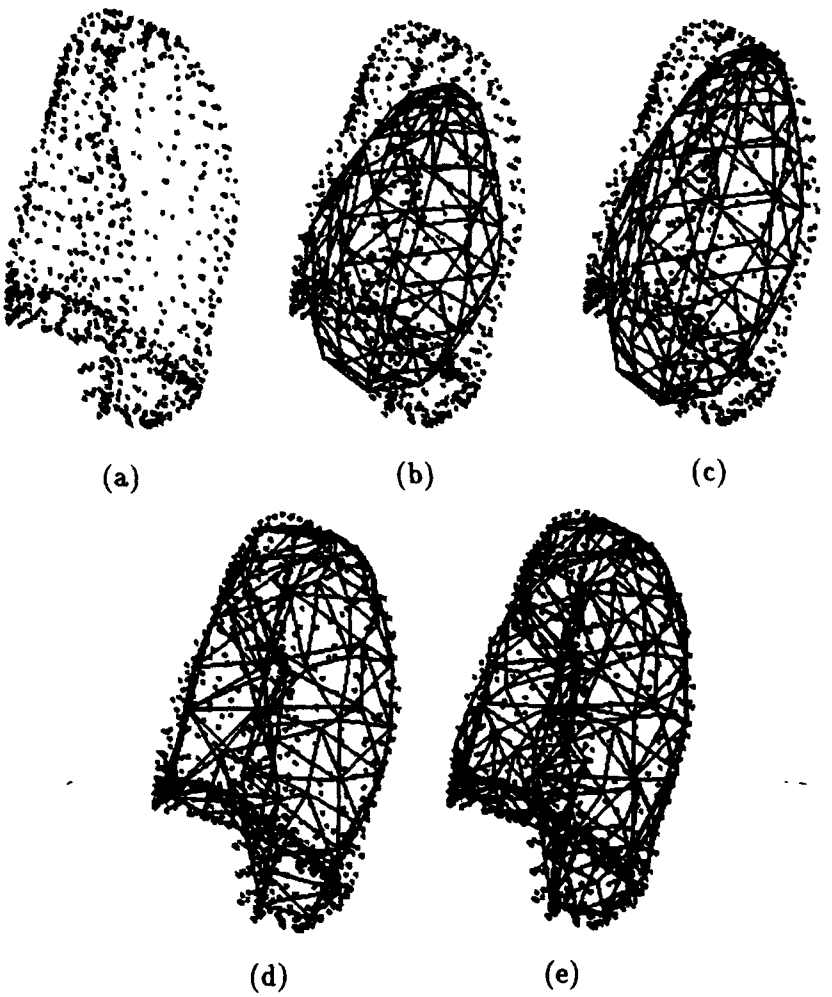


Figure 6: Fitting of model to data of a human lung. (a) input data of a human lung (left part), (b) model initialization, (c) model fitted to data with only global deformations, (d) model fitted to data with global and local deformations, (e) model fitted to data after three levels of local subdivision.

model	sub1	sub2	sub3	sub4	nosub1	nosub2
# of nodes	66	129	168	243	256	627
# of elements	128	254	332	482	512	1250
# of iterations	11401	16928	21298	27303	20901	22251
$mean_{error}$	0.081739	0.023317	0.013317	0.008100	0.019040	0.011834
max_{error}	1.351201	0.371757	0.318139	0.104073	0.281506	0.247791
σ_{error}	0.162214	0.038362	0.017991	0.010340	0.034987	0.026694
σ^2_{error}	0.026313	0.001472	0.000324	0.000107	0.001224	0.000713

Table 1: Error estimates of fitting of the model to 857 data points of a human lung.

fitting errors for each fitting of the model. For this application, for each level of subdivision, we allowed fitting until the maximum model movement measure becomes less than 0.00005. Then we subdivided the model elements as explained in the previous sections. In selecting elements for subdivision, we used a distance threshold of $\tau_d = 0.3$. We started fitting the model with 66 nodes and 128 elments while the number of input data points was 857. At the first level of subdivision, 87 elements were selected for subdivision, and the model resulted in a total of 129 nodes and 254 elements. At the second level of subdivision, 50 elements were selected and the subdivision process produced a total of 168 nodes and 332 elements. At the third level, 77 elements were selected, and the subdivision process generated a total of 243 nodes and 482 elements.

We also fitted a model with a number of elements comparable to the final subdivided model and measured the fitting error. This model contained 256 nodes and 512 elements. As shown in the table, the mean and maximum errors derived using this model notably exceed those of the subdivided model. Table 1 also shows the error measures obtained by fitting the model with 627 nodes and 1250 elements. Through this example it is apparent that our locally adaptive subdivision significantly improves the model fitting.

In Fig. 7, we fit a deformable model with 47 nodes to 470 3D data points sampled from a human upper left arm. The input image is given in Fig. 7(a). The local deformation stiffness parameters of the model were $\omega_0 = 0.1$ and $\omega_1 = 0.1$. The initial model was an ellipsoid ($\mathbf{q}_2 = (8.0, 0.21, 0.21, 0.8, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0)^T$) and the force strength parameter was $\beta = 20.0$.

Table 2 shows error estimation collected after fitting the deformable models to the above input data. We again show the number of iterations, the mean value, the maximum value, the standard deviation and the variance of the fitting error for each model. For the model with subdivision, for each level of subdivision, we allowed the model to fit until the maximum model movement measure becomes less than 0.00005. Then we subdivided the model elements as explained in the previous sections. In selecting elements for subdivision, we used a distance threshold of $\tau_d = 0.2$. The number of input data points was 470. We started fitting

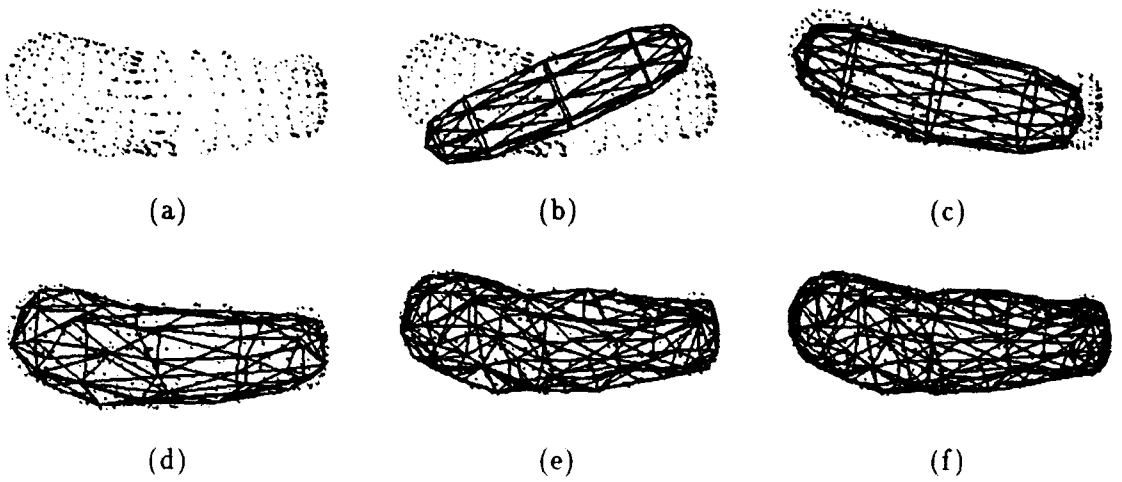


Figure 7: Fitting of model to data of a human arm. (a) input data of a human arm (left part), (b) model initialization, (c) model fitted to data with only global deformations, (d) model fitted to data with global and local deformations, (e) model fitted to data after two levels of local subdivision, (f) model fitted to data after three levels of local subdivision.

model	sub1	sub2	sub3	sub4	nosub1
# of nodes	47	107	241	343	402
# of elements	90	210	478	682	800
# of iterations	4351	9249	14548	25546	128144
$mean_{error}$	0.118530	0.086852	0.007490	0.001484	0.002796
max_{error}	4.399110	0.927110	0.402416	0.047835	0.203244
σ_{error}	0.263431	0.096933	0.022171	0.003416	0.010973
σ^2_{error}	0.69396	0.009396	0 000492	0.000012	0.000120

Table 2: Error estimates in fitting of the model to 470 data points of a human upper arm.

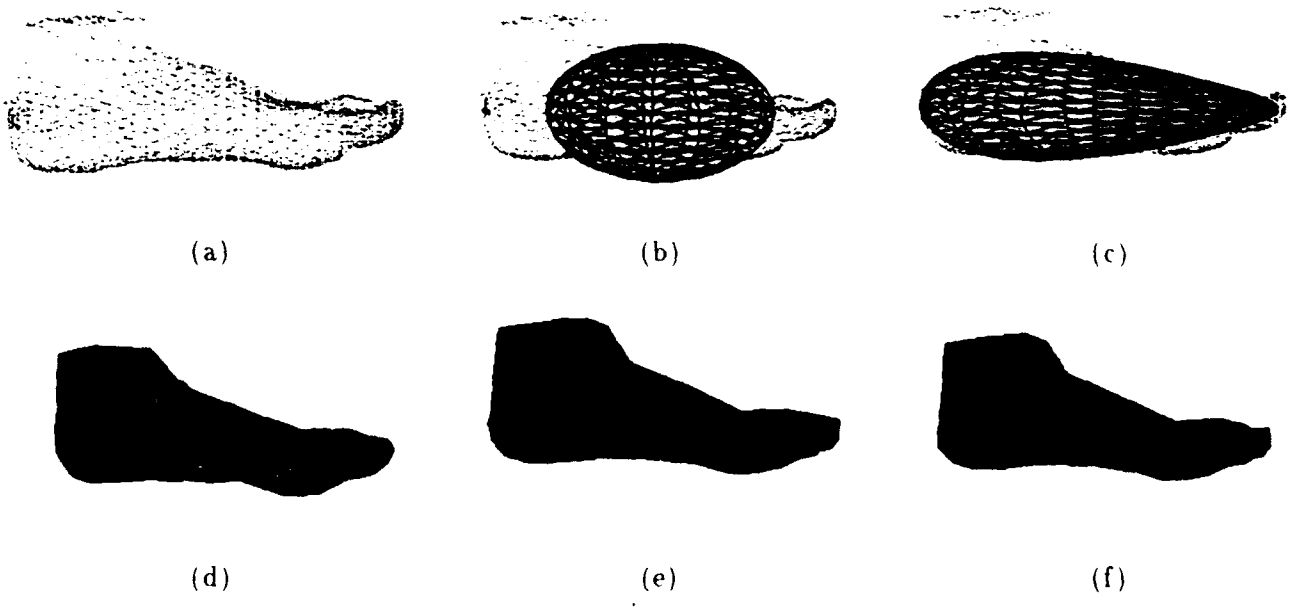
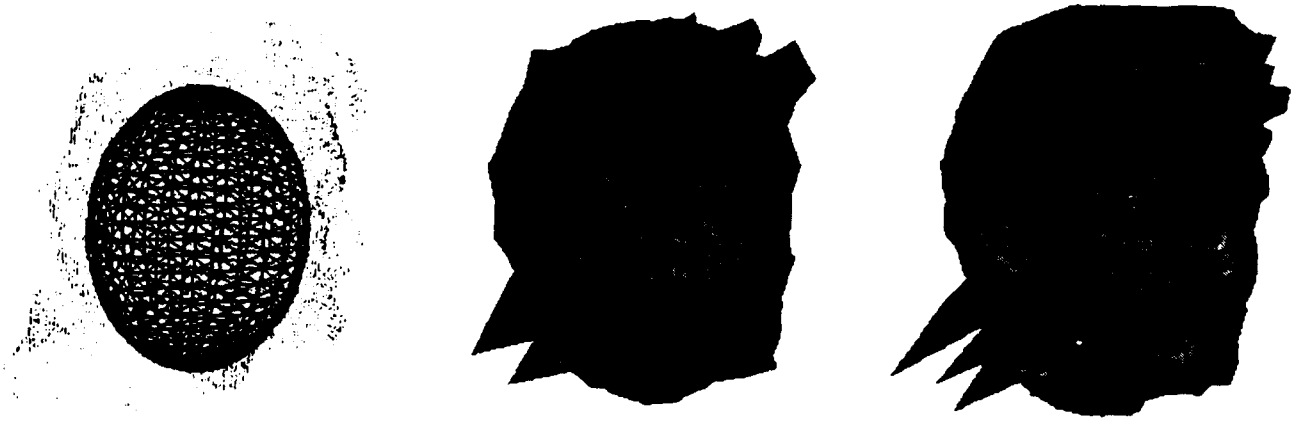


Figure 8: Fitting of model to foot data. (a) Foot data, (b) Model Initialization, (c) Intermediate step of model fitting to the data with apparent global deformations, (d) Model fitted to data without local subdivision, (e) Model fitted to data after one level of local subdivision, (f) Model fitted to data after four levels of local subdivision.

of the model with 47 nodes and 90 elements. At the first level of subdivision, 86 elements were selected for subdivision, and the model resulted in a total of 107 nodes and 210 elements. At the second level of subdivision, 173 elements were selected, and the subdivision process produced a total of 241 nodes and 478 elements. At the third level, 34 elements were selected, and the subdivision process generated a total of 343 nodes and 682 elements. We also fitted a model with a number of elements comparable to the number of elements in the final subdivided model and measured the fitting error. This model contained 402 nodes and 800 elements.

Fig. 7(a) shows a view of the range data. Fig. 7(b) shows the initial model. Fig. 7(c) shows the model after global deformations. After global deformations, we obtained scaling parameters of $\mathbf{q}_s = (8.0, 0.238, 0.304, 0.817)^T$ and quaternion of $\mathbf{q}_\theta = (40.621, -0.694, -0.081, 0.095)^T$. Bending deformation was applied to result in $\mathbf{q}_b = (0.371, 0.105, 0.670)^T$. Fig. 7(d) shows the model after local deformations. Fig. 7(e) shows the intermediate model after two levels of local subdivision. Fig. 7(f) shows the intermediate model after three levels of local subdivision.

In the following two experiments we use range datapoints obtained from the Cyberware, Inc., 3D digitizer. In Fig. 8 we fit a deformable model with 227 nodes initially to 3825 3D range datapoints obtained from a mannequin foot. The local deformation stiffness parameters of the model were $w_{00} = 0.5$, $w_{01} = 0.5$, $w_{10} = 0.5$, $w_{02} = 0.1$, $w_{11} = 0.1$ and $w_{20} = 0.1$. The initial model was an ellipsoid ($\mathbf{q}_s = (2.3, 0.3, 0.5, 0.3, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)^T$) and the force strength parameter was $\beta = 20.0$. Fig. 8(a) shows the given foot data. Fig. 8(b) shows a view of the range data and the initial model, while Fig. 8(c) shows an intermediate step in the fitting of the model to the data where the global deformations are apparent. Fig. 8(d) shows the model fitted to the data without local subdivision, Fig. 8(e) shows the



(a) (b) (c)

Figure 9: Fitting of model to head data. (a) Model Initialization. (b) Model fitted to data without local subdivision, (c) Model fitted to data after five levels of local subdivision.

model fitted to the data after one level of local subdivision, while Fig. 8(f) shows the final model fitted to the data after four levels of local subdivision. The new final number of model nodes is 640, which is significantly smaller than the number of given datapoints.

In Fig. 9 we fit a deformable model with 627 nodes initially to 5070 3D range datapoints obtained from a head. The local deformation stiffness parameters of the model were $w_{00} = 0.005$, $w_{01} = 0.001$, $w_{10} = 0.001$, $w_{02} = 0.00001$, $w_{11} = 0.00001$ and $w_{20} = 0.00001$. The initial model was an ellipsoid ($\mathbf{q}_s = (2.5, 0.5, 0.5, 0.65, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)^T$) and the force strength parameter was $\beta = 20.0$. Fig. 9(a) shows a view of the range data and the initial model. Fig. 9(b) shows the model fitted to the data without local subdivision, Fig. 9(c) shows the final model fitted to the data after five levels of local subdivision. The improvement in the shape with respect to the level of subdivision is obvious. The new final number of model nodes is 1597, which is significantly smaller than the number of given datapoints.

In Fig. 10 we fit a deformable model with 289 nodes initially to 2929 3D range datapoints of a mask obtained from the NRCC Image database. The local deformation stiffness parameters of the model were $w_{00} = 0.005$, $w_{01} = 0.001$, $w_{10} = 0.001$, $w_{02} = 0.0001$, $w_{11} = 0.0001$ and $w_{20} = 0.0001$. The initial model was an ellipsoid ($\mathbf{q}_s = (2.7, 0.3, 0.4, 0.55, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0)^T$) and the force strength parameter was $\beta = 10.0$. Fig. 10(a) shows a view of the range data and the initial model. Fig. 10(b) shows the final model fitted to the data after two levels of local subdivision. The new final number of model nodes is 526, which is again significantly smaller than the number of given datapoints.

In this experiment we fit a deformable model with 47 nodes to 1269 3D data points defining a human head. The input data were obtained from Viewpoint, Inc. The local deformation stiffness parameters of the model were $\omega_0 = 0.05$ and $\omega_1 = 0.05$. The initial model was an ellipsoid ($\mathbf{q}_2 = (7.0, 0.5, 0.5, 0.6, 1.0, 1.0, 0.0, -0.3, 0.0, 0.0, 1.0)^T$) and the force strength parameter was $\beta = 1.0$.

Fig. 11(a) shows a view of the range data. Fig. 11(b) shows the initial model. Fig. 11(c) shows the model after global deformations. Fig. 11(d) shows the model after local deformations. Fig. 11(e) shows the intermediate model after two levels of local subdivision. Fig. 11(f) shows the intermediate model after

(a) (b)

Figure 10: Fitting of model to mask data. (a) Model Initialization, (b) Model fitted to the data after two levels of local subdivision.

four levels of local subdivision.

Fig. 12 (a) and (b) respectively show a front view and a back view of human body figures displayed at three different levels of detail. The human body figure consists of 15 parts: head, torso, lower torso, 3 parts for each arm, 3 parts for each leg. For coarser levels of detail, approximations of each body part were obtained as described in the above experiments. The numbers of polygons used at each level of representation were 18155, 7292, and 2260 respectively, and the numbers of nodes were 18005, 3696, and 1180 respectively.

6 Conclusions

We have developed a new technique that allows the local adaptive subdivision of the finite elements representing the local deformations of our deformable models. Our algorithm ensures that during subdivision the desirable finite element mesh generation properties of conformity, non-degeneracy and smoothness are maintained. In conjunction with the use of our new force assignment technique from datapoints to model points, we not only represent more accurately an object surface, but also more efficiently because new model nodes are added only when necessary in a local fashion. Finally, using our new locally adaptive finite element technique and the global deformations of our models we can achieve a smooth hierarchical shape representation.

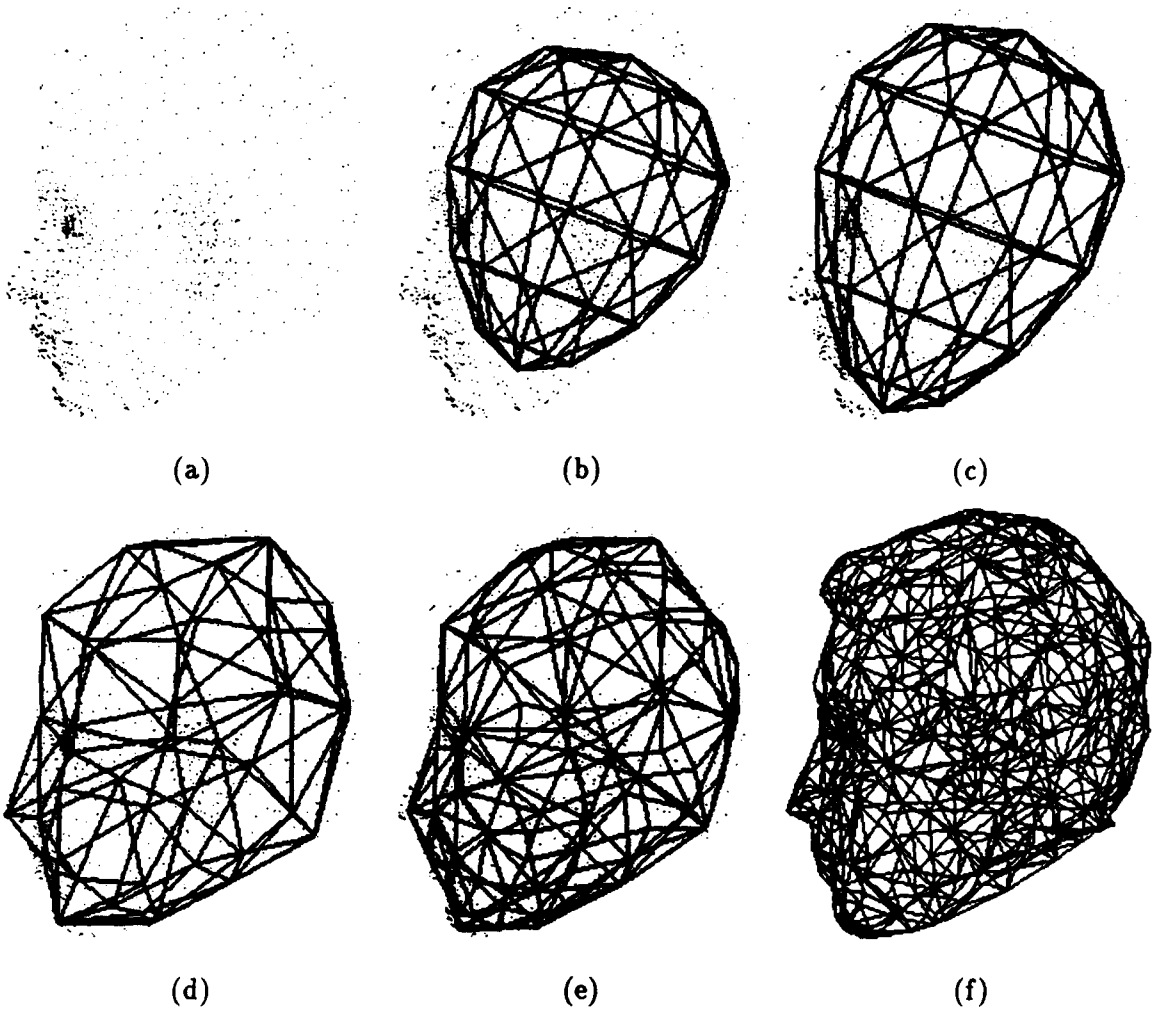
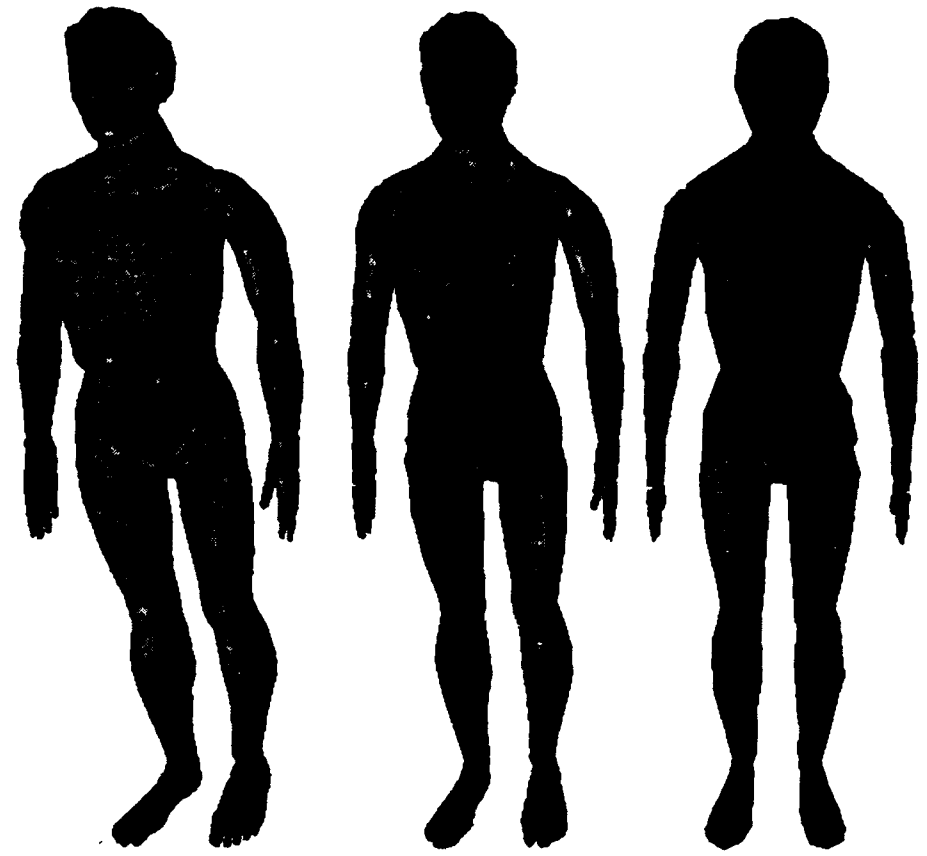
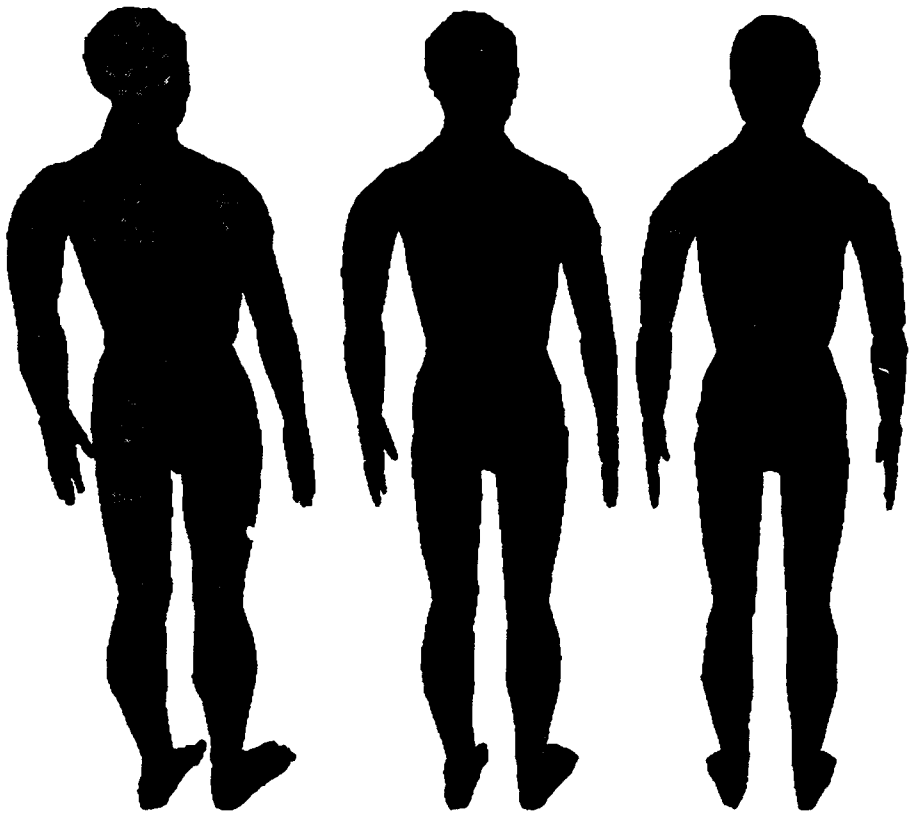


Figure 11: Fitting of the model to input data of a human head. (a) input data, (b) the initial model, (c) the model after global deformations, (d) the model after local deformations, (e) the model after two levels of subdivision, (f) the model after four levels of subdivision.



(a)



(b)

Figure 12: A human body displayed at three different levels of detail: (a) the front view, (b) the back view.

References

[1] A. Barr. Superquadrics and Angle-Preserving Transformations. *IEEE Computer Graphics and Applications*, 1:11–23, 1981.

[2] J. H. Bramble, J. E. Pasciak and J. Xu. The Analysis of Multigrid Algorithms with Nonnested Spaces of Noninherited Quadratic Forms. Mathematics of Computation 56(193), pp. 1-34, 1991.

[3] L. Cohen and I. Cohen. Finite Element Methods for Active Contour Models and Balloons for 2D and 3D Images. PAMI # 91-11-19.

[4] I. Cohen, L. Cohen nd N. Ayache. Using Deformable Surfaces to Segment 3D Images and Infer Differential Structure. CVGIP: Image Understanding, 56(2), pp. 242-263, 1992.

[5] W. C. Huang, and D. Golgof, Adaptive-Size Physically-Based Models for Nonrigid Motion Analysis. Proc. IEEE Computer Vision and Pattern Recognition Conference (CVPR'92), pp. 833–835, Champaign, Illinois, June 1992.

[6] D. Metaxas, and D. Terzopoulos, Constrained Deformable Superquadrics and Nonrigid Motion Tracking, Proc. IEEE Computer Vision and Pattern Recognition Conference (CVPR'91), Hawaii, pp. 337–343, June 1991.

[7] D. Metaxas, and D. Terzopoulos, Shape and Nonrigid Motion Estimation Through Physics-Based Synthesis, IEEE Patt. Anal. and Mach. Intell., 1992, in press.

[8] D. Metaxas, Physics-Based Modeling of Nonrigid Objects for Vision a..d Graphics. Ph.D. Thesis, Department of Computer Scinece, University of Toronto, 1992.

[9] T. McInerney. Finite Element Techniques for Fitting Deformable Models to 3D Data. *M.Sc. Thesis, Dept. of Computer Science, University of Toronto, Toronto, CA*, 1992.

[10] A. Pentland and B. Horowitz. Recovery of Non-rigid Motion and Structure. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7):730–742, 1991.

[11] M. C. Rivara, Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, International Journal for Numerical Methods in Engineering, 20, pp. 745–756, 1984.

[12] I. G. Rosenberg, and F. Stenger, A lower bound on the angles of triangles constructed by bisecting the longest side,*Math. Comp.*, 29, pp. 390–395, 1975.

[13] M. Stynes, On fast convergence of the bisection method for all triangles, *Math. Comp.*, 35, pp. 1195–1201, 1980.

[14] H. Tanaka anf F. Kishino. Adaptive Mesh Generation for Surface Reconstruction: Parallel Hierarchical Triangulation Without Discontinuities. Proc. CVPR'93, pp. 88-94, NY, 1993.

[15] D. Terzopoulos and D. Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(7):703–714, 1991. See also Proc. Third International Conference on Computer Vision (ICCV'90), Osaka, Japan, Dec. 1990, pp. 606-615.

[16] D. Terzopoulos and A. Witkin. Physically Based Models with Rigid and Deformable Components. *IEEE Computer Graphics and Applications*, 8(6):41–51, 1988.

[17] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on Deformable Models: Recovering 3D Shape and Nonrigid motion. *Artificial Intelligence*, 36(1):91–123, 1988.

[18] M. Vasilescu, and D. Terzopoulos, Adaptive Meshes and Shells: Irregular Triangulation, Discontinuities, and Hierarchical Subdivision, Proc. IEEE Computer Vision and Pattern Recognition Conference (CVPR'92), pp. 829–832, Champaign, Illinois, June 1992.

[19] B.C. Vemuri and A. Radisavljevic. From Global to Local, a Continuum of Shape Models with Fractal Priors. Proc. CVPR'93, pp. 307-313, NY, 1993.

[20] O. Zienkiewicz. *The Finite Element Method.* McGraw-Hill, 1977.

D A 3-D Model of Tongue Movements Using Soft Object Techniques: Catherine Pelachaud, Chin Seah, C.W.A.M. van Overveld

A 3-D Model of Tongue Movements Using Soft Object Techniques

Catherine Pelachaud, Chin Seah, C.W.A.M. van Overveld

1 Abstract

A geometric and kinematic model for describing the global shape and the predominant motions of the human tongue, to be applied in computer animation, is discussed. The model consists of a spatial configuration of moving points that form the vertices of a mesh of 9 3-D triangles. These triangles are interpreted as charge centres (the so called skeleton) for a potential field, and the surface of the tongue is modelled as an equi-potential surface of this field. In turn, this surface is approximated by a triangular mesh prior to rendering. As to the motion of the skeleton, precautions are taken in order to achieve (approximate) volume conservation; the computation of the triangular mesh describing the surface of the tongue implements penetration avoidance with respect to the palate. Further, the motions of the skeleton derive from a formal speech model which also controls the motion of the lips to arrive at a visualy plausible speech synchronous mouth model.

2 Introduction

In this paper, a simple tool is discussed to model the shape of a human tongue. This tool also supports simulated tongue movements during speech production, useful in the context of computer animation. In real life, the tongue plays an important role in speech production. Some phonemic elements are not differentiated by their corresponding lip shapes; rather they are distinguished by the movement of the tongue (for example /d/, /t/...). Even though only a small portion of the tongue is visible during normal speech, taking the tongue shape into account will enhance the visual plausibility of a computer graphics facial animation system. A geometric tongue model, however, is far from trivial: indeed, the tongue is a complex and flexible organ with highly articulated and irregular motions. In most facial animation system, tongue movement is not considered, or if so it is over simplified. In most of the cases it is represented by a parallelepiped that can move inward, outward, upward, and downward [12], [3], [17], [14]. We propose to model the tongue based on the soft object technique of [27]. This technique assumes a so called skeleton, comprising of few geometric primitives (in our case 9 triangles) that serves as a charge distribution causing a spatial potential field. The modelled soft object is an equi-potential surface defined by this field. Modifying

the skeleton will modify the equi-potential surface, i.e. the soft object. Since the skeleton has only few degrees of freedom, defining the behavior over time of the skeleton is a convenient way to define the behavior over time of the resulting complex shape. We propose an interactive tool to model the shape of the skeleton; moreover we propose an algorithm to compute the soft object which implements penetration avoidance such that the tongue stays within the palate at each frame while the volume is approximately constant.

In the next section we explain our model and we discuss how the primitives for the model are built. We also summarize briefly our implementation of the soft object technique. The user is referred to [24] for more details on the construction of a triangle mesh to represent the equi-potential surface. In the subsequent section we describe our penetration avoidance algorithm. Finally we show how we compute tongue shapes during speech production. Lip shapes are also computed and coarticulation effects are taken into account to produce the final animation.

3 tongue definition

The human tongue plays a significant part in speech production. Sounds are differentiated, among other factors, by the position of the tongue related to the palate, and by the curvature and contraction of the tongue.

The tongue is a highly flexible organ. It comprises of muscles, fat and connective tissue [21]. Longitudinal and transverse muscles interleave each other. Their contraction patterns determine the direction of the tongue deformation. The contraction of longitudinal muscle will shorten and draw back the tongue while the contraction of the other group of muscles will flatten and extend it [21]. Moreover the tongue can be bent, twist and tensed [20].

4 Tongue Modeling

Our goal is to find a compromise between a highly flexible structure with very complex movements and a simple representation made up from few primitives, each with few parameters. In this respect, the soft object technique seems to be a promising approach. Few primitives (9 triangles) define the model. We present first how we form the skeleton for the soft object. Then we discuss a tool to help create different skeleton shapes. This tool relates the geometric parameters from the skeleton

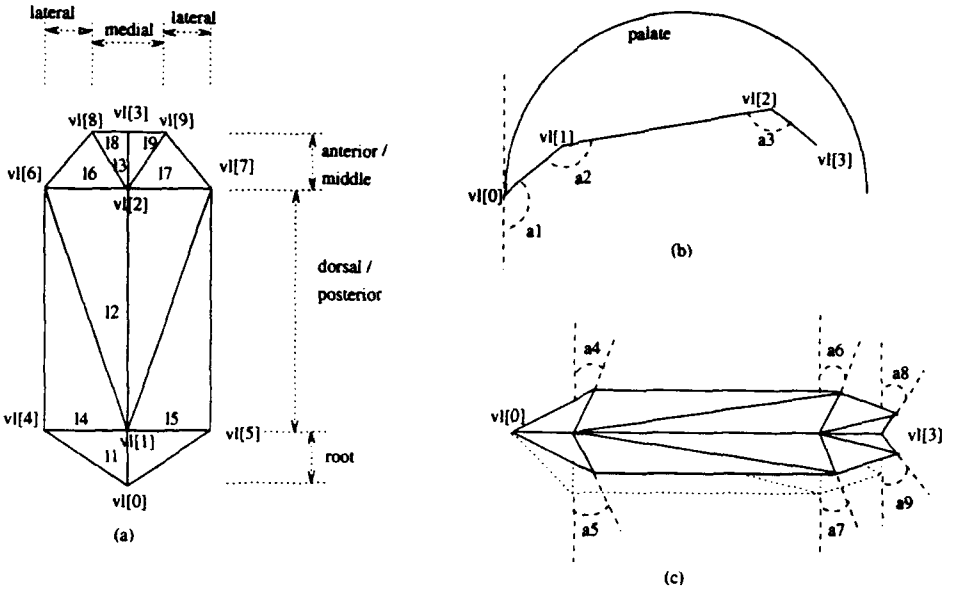


Figure 1: Skeleton of the tongue

(i.e. the locations of the vertices of the 9 triangles) to what will be called shape parameters. Each shape parameter implements a meaningful shape attribute of the tongue; each shape parameter can be modified interactively. Finally we explain how the final shape of the tongue is computed from the skeleton; to this aim, the soft object technique will be explained briefly.

4.1 3-D Model

In [21], Maureen Stone proposed a 3-D model of the tongue. She defined 5 segments in the coronal plane – one medial and two laterals (on each side of the median) – and 5 segments in the sagittal plane – root, posterior, dorsal, middle and anterior. Our model can be compared with Maureen Stone's 3-D model. On the one hand we want to be able to model an asymmetric tongue shape and on the other hand we want to keep the number of degrees of freedom possibly low. Therefore we retain only 3 segments in the sagittal plane and 3 segments in the coronal plane (see figure 1).

By moving points $vl[1]$ and $vl[2]$ along the median, they will represent respectively the anterior/middle and the dorsal/posterior degrees of freedom. In normal speech, the anterior and middle segments are never independent characteristics of a tongue shape simultaneously (similar for dorsal and posterior), so these don't have to occur as independent shape parameters. Therefore, in our model the remaining segments in the coronal plane are one medial and one lateral segment. Bent,

Figure 2: Two examples of tongue shapes

twisted, and curved shapes can still be represented with our model as well as asymmetric shapes, but the "groove" shape can not be modelled independently, anymore. Nevertheless, since we aim at modelling normal speech, where only a small portion of the tongue contributes to the visual appearance of the mouth, this approaximation turns out to be sufficiently versatile for modelling tongue shape during speech production.

4.2 Geometric Representation

The tongue model consists of 9 triangles. The median is divided into three parts. The two middle points $vl[1]$ and $vl[2]$ can move along the median. A tool has been developed to modify interactively and independently each shape parameter of the model; these shape parameters are: the lengths of the edges l_i forming the median and the angles a_i between these edges. Each modification creates a new tongue shape (figure 2). By rotating the segments in the sagittal plane the tongue can be made to bend or roll. The external points $vl[i]$ can be moved by rotating the edges in the coronal plane: the tongue can be made to twist or take a U-shape.

Modifying the lengths of the edges will modify the tongue surface: the tongue can be made to compress, stretch, narrow, or flatten.

The relations between the points of the tongue skeleton (geometric parameters) and the shape

parameters are (please also refer to figure 1 for the meaning of the shape parameters): vl are the points, l_i are the size of the segments and a_i are the angles):

$$
\begin{aligned}
vl[0].x &= x_0; \\
vl[0].y &= y_0; \\
vl[0].z &= z_0; \\
vl[1].x &= vl[0].x + (l_1 * \sin(a_1)); \\
vl[1].y &= vl[0].y - (l_1 * \cos(a_1)); \\
vl[1].z &= vl[0].z; \\
vl[2].x &= vl[1].x - (l_2 * \sin(a_2)); \\
vl[2].y &= vl[1].y + (l_2 * \cos(a_2)); \\
vl[2].z &= vl[1].z; \\
vl[3].x &= vl[2].x + (l_3 * \sin(a_3)); \\
vl[3].y &= vl[2].y - (l_3 * \cos(a_3)); \\
vl[3].z &= vl[2].z; \\
vl[4].z &= vl[1].z - (l_4 * \cos(a_1)); \\
vl[4].x &= vl[1].x + (l_4 * \sin(a_1)); \\
vl[4].y &= vl[1].y; \\
vl[5].x &= vl[1].x + (l_5 * \sin(a_2)); \\
vl[5].y &= vl[1].y; \\
vl[5].z &= vl[1].z + (l_5 * \cos(a_2)); \\
vl[6].x &= vl[2].x + (l_6 * \sin(a_3)); \\
vl[6].y &= vl[2].y; \\
vl[6].z &= vl[2].z - (l_6 * \cos(a_3)); \\
vl[7].x &= vl[2].x + (l_7 * \sin(a_4)); \\
vl[7].y &= vl[2].y; \\
vl[7].z &= vl[2].z + (l_7 * \cos(a_4)); \\
vl[8].x &= vl[3].x + (l_8 * \sin(a_5)); \\
vl[8].y &= vl[3].y; \\
vl[8].z &= vl[3].z - (l_8 * \cos(a_5)); \\
vl[9].x &= vl[3].x + (l_9 * \sin(a_6)); \\
vl[9].y &= vl[3].y; \\
vl[9].z &= vl[3].z + (l_9 * \cos(a_6));
\end{aligned}
$$

4.3 The Soft Object Technique

Equi-potential surfaces are a sub-class of implicit functions. Among other things, they can serve to model soft objects. Equi-potential surfaces are expensive to render directly (e.g. using ray tracing); rather, they should be converted into a triangle mesh prior to rendering. In [24], a method is proposed to convert an equi-potential surface into a triangle mesh in such a way that the triangle

shapes adapt to the local curvature of the equi-potential surface: relatively flat areas give rise to large triangles whereas small triangles occur in strongly curved regions; moreover, isotropically curved surface regions translate into nearly equilateral triangles and highly anisotropically curved regions give very acute triangles.

In order to implement this, the notions of an *acceptable surface* and of *acceptable edges* are introduced. An acceptable surface is a surface where for each two points, a and b, the angle between the normals in a and b does not exceed a constant factor β times $|a-b|$. The value of β relates to the maximal curvature of the surface. For an edge ab to be acceptable, its length $|a-b|$ should not exceed a given threshold L_{max} and the angle between the normals in the points a and b should not exceed a given threshold value α. Given β, α, and L_{max} quantitative estimates for the maximal deviation of the surface and the triangular mesh approximation can be derived (see [24]).

Summarising, the tessalation is characterised by:

- the surface is given by $f(r) = 0, r \in \Re^3$
- a cord (edge of a triangle) is a tuple (a, b, na, nb) where

$$f(a) = f(b) = 0 \quad \text{and} \quad na = \nabla f(a) \quad \text{and} \quad nb = \nabla f(b)$$

- the surface is called acceptable iff for every cord ab on the surface

$$\angle(na, nb) \leq \beta \mid a - b \mid$$

- a cord is called acceptable iff

$$\angle(na, nb) \leq \alpha \quad \text{and} \quad \mid a - b \mid \leq L_{max}$$

- a triangle consisting of three cords is acceptable iff all three cords are acceptable

In this case, $f(r)$ is a potential field, caused by a set of point charges. Each triangle contributes one point charge; in order to compute the potential in point r in space, the point charge is located in the point R within the triangle, closest to r. Such a point charge is represented by a tuple (R, ρ) where R is the in 3-D space and ρ is its charge. The potential due to this point charge in point r is

$$f(r) = \frac{\rho}{\mid r - R \mid}$$

If two triangles share an edge, the resulting potential is twice as high near this edge which results in unwanted bulging of the equi-potential field. This is remedied by adding line charges located at the common edges with negative contributions. In turn, this would cause over compensation near the vertices where common edges meet, so we also have to add positive point charges in the common vertices.

For all triangles, lines and vertices, the combined equi-potential surface is

$$S = \{r \in \Re^3 \mid \sum_i \frac{\rho_i}{\mid r - R_i \mid} = V_0\}$$

The algorithm discussed in detail in [24] guarantees that the vertices of the adaptive triangular mesh approximating S are on $f(r) = V_0$; that a closed surface results, and that the surface is tessalated by acceptable chords only.

To assure that vertices lay on the surface, initially the value of V_0 is set to a value close to 0. As a result, S will be very large and nearly spherical. A sphere-shaped surface is straightforwardly triangulated, and adaptiveness does not matter since the curvature is the same everywhere. Next, V_0 is increased slightly. The surface S shrinks and may become slighly more involved. Since the vertices only have to move little, however, a linearisation of the expression for $f(r)$ is sufficiently accurate to compute new locations of the vertices. The acceptability criterion is checked for all edges; if an edge is not acceptable it is split. In this manner, the value of V_0 is increased in several steps until it reaches the value for which the final shape of the equi-potential surface is defined. This stepwise approach assures that underway the vertices of the triangle mesh stay on the (shrinking) surface S and the repeated checking of the acceptability criterion guarantees that an adaptive triangulation results.

In order to achieve a closed surface, the starting polyhedron is chosen to be closed; e.g. one can take a tetrahedron which is the simplest closed triangular mesh.

5 Animation

Animating the skeleton over time, given an input text, is achieved by outputting for each phonemic item the values of all the shape parameters (the edge lengths of the median and the angles between these edges) defining the skeleton. Thus a tongue skeleton is obtained for every key-frame. Since during the construction of a soft object, the number of vertices in the triangular mesh is not

necessary the same for all shapes of the skeleton, interpolating the resulting meshes is in general not well possible. Instead, the animation is obtained by interpolating between tongue skeletons; so to each frame corresponds a tongue skeleton and the soft object algorithm computes for every frame the final tongue shape.

5.1 Compressibility and Velocity

In real life, compressibility is an important feature of the tongue. The tongue does not extend or retract uniformly along its surface. Each segment can be compressed or extended independently. The pattern of compression and expansion varies over the tongue.

The middle segment has a tendency to be more compressed than the other segments. Consonants and vowels show different patterns of compression and retraction. During the production of consonant, the tongue tries to reach the palate and shows greater change of position. During the production of vowels, the tongue has the tendency to compress more in order to open the vocal tract. For vowels the degree of compressibility is mainly a function of tongue height.

Some differences between segments occurs also in the timing of the tongue movement. Some points arrive earlier followed by the other points: each segment has its own characteristic velocity [20].

Regardless of context, the vowel expansion and compression patterns vary roughly as a function of tongue height. The higher vowels, /i/ and /o/, cause the anterior segment to become compressed and retracted whereas the dorsal segment moves upward. For /a/, the middle and dorsal segments are compressed.

Finally, for vowels the tongue is more compressed than for consonants. This reflects the fact that the tongue displaces farther for the consonants to contact the palate and it retracts for the vowels to open the vocal tract.

6 Penetration Avoidance

When the tongue moves we have to check that it does not penetrate the palate. The palate is modeled as a semi-sphere and a finite, strip-shaped plane. The penetration avoidance algorithm works in two passes. The first pass takes place at the skeleton level; the second pass takes place when the triangle mesh representing the soft object is computed. A possible penetration is detected easier and faster during the first pass since here checking involves fewer points. This step assures

the tongue skeleton to be within a given sphere-and-plane configuration, sufficiently small within the palate (to be called the virtual palate) to guarantee that the associated equi-potential surface does not penetrate the true palate; also, in the first pass the area of the triangles of the tongue skeleton is kept approximately constant which implements approximate volume conservation of the tongue as a whole. The second pass corrects for possible penetrations of the resulting equi-potential surface with the true palate.

The algorithm works essentially the same in both passes. First, a possible penetration is checked, either with the skeleton and the virtual palate or the equi-potential surface and the true palate. If there is no penetration, the algorithm terminates. If there is a penetration, the penetrating points (of the skeleton or of the triangular mesh representing the equi-potential surface) are moved back inside the (virtual) palate. In order to preserve the volume, in pass one the algorithm assures that

- the length of a segment with a penetrating extreme remains constant;
- if a segment of the sagittal plane is compressed (respectively expanded), the other segments of the coronal plane expand (respectively compress) to compensate.

6.1 First pass

Here we describe how to check for penetration of the virtual palate, modelled using a sphere and a planar strip, by the skeleton. Also, the criteria to be used for preserving volume of the tongue are discussed.

6.1.1 Penetration of a Sphere

The first pass takes place when computing the shapes of the skeleton over time. The program checks if, for each frame, each vertex of the tongue skeleton is within the virtual palate. The first check is within the sphere part of the virtual palate. The condition for penetrating the sphere is:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 > r^2,$$

where (x, y, z) are the coordinates of a vertex, (x_0, y_0, z_0) are the coordinates of the center of the sphere, and r is its radius.

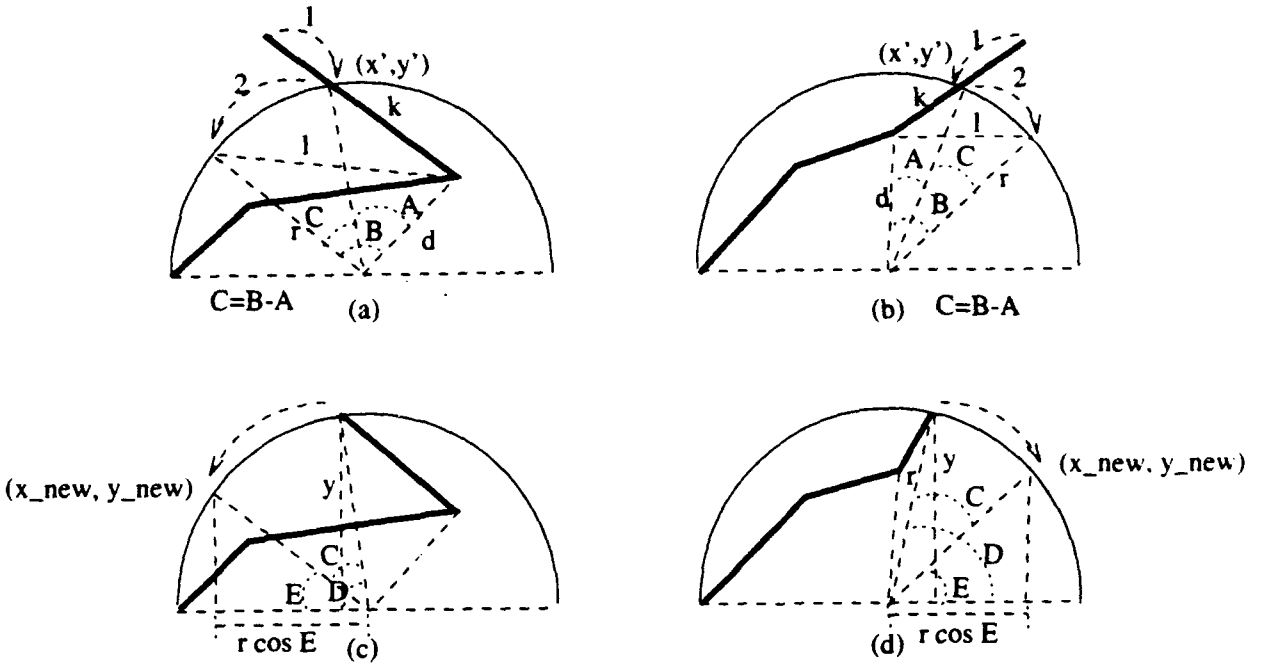


Figure 3: Collision with a sphere

If a vertex penetrates the sphere, its new position (x', y', z') is found by projecting onto the sphere:

$$x' = x_0 + ((\frac{r}{d}) * (x - x_0)),$$

where $d = |(x, y, z) - (x_0, y_0, z_0)|$ (see figure 3 (a) and (b)). Similar formulas hold for y and z coordinates.

Moving a vertex as described above would change the length l of the incident edges in the skeleton. Since these lengths are shape parameters that carry significance for the current tongue shape, however, they should be invariant. Therefore, instead of merely translating the single affected vertex, the algorithm rotates the incident segment instead. Firstly, we need to determine the angle to rotate which is equal to the angle between the vertex that moved and its neighbouring vertex:

$$B = \arccos(\frac{r^2 + d^2 - l^2}{2 * r * d})$$

$$A = \arccos(\frac{r^2 + d^2 - k^2}{2 * r * d})$$

Therefore, the angle of rotation is $C = B - A$ (see figure 3 (a) and (b)).

Next, we need to either rotate left or right along the arc of the palate depending on where the vertex was before. In case of a left rotation along the arc of the palate, again two cases need to be considered (see figure 3 (c) and (d)). Two angles need to be computed:

$$D = \arcsin(\frac{y}{r}) \quad \text{and} \quad E = D - C$$

The new coordinates of the vertex will be finally:

$$x_{new} = r - (r * \cos(E)) \quad \text{and} \quad y_{new} = r * \sin(D))$$

The case of right rotation is similar. This completes the penetration check with the sphere.

6.1.2 Penetration of a Plane

If a vertex crosses the plane (for simplicity, the coordinate system is assumed to be perpendicular to this plane), the algorithm takes the following steps:

- compute the normal distance *dist* of the vertex from the plane:

$$dist = \frac{p_0 * x + p_1 * y + p_2}{\sqrt{p_0^2 + p_1^2}},$$

  where p_0, p_1, p_2 define the plane.

- next the vertex is moved perpendicularly onto the surface. This is done iteratively, i.e. by moving the point along the normal in several steps:

$$x_{new} = x - (\frac{dist * p_0}{step}),$$

  where again x_{new} is the new x coordinate and *step* is the step of the iteration. We do the same for y coordinate. At each step, the new vertex is checked if it is inside the virtual palate. If it is the algorithm terminates; otherwise it reiterates : the vertex moves along the normal and the check is done once more.

6.1.3 Expand / Contract

In order to conserve the tongue volume, the change in length in one direction should be compensated by a change in the other direction. Using the soft object technique allows producing objects from

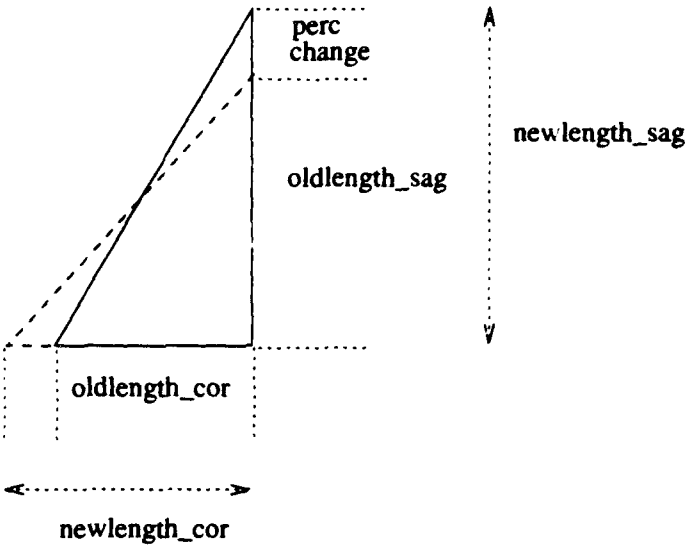


Figure 4: Extansion / Contraction of the tongue

few primitives and to define their animation in an intuitive way. Nevertheless, it does not guarantee volume conservation of the equi-potential surface. For our purpose we ignore these volume changes. At the skeleton level, however, we can strive for area conservation of the triangles forming the skeleton. The penetration avoidance algorithm modifies segment lengths in the sagittal plane; so the algorithm should adjust accordingly the segment lengths in the coronal plane.

In the coronal plane the skeletal frame is expanded and contracted as follows. After detecting a penetration of the virtual palate in the saggital plane, the algorithm compares the segment's lengths and computes the ratio of the change in length:

$$ratio_{change} = \frac{(oldlength_{sag} - newlength_{sag})}{oldlength_{sag}}$$

Then in the coronal plane, the corresponding sides of the segment are extended or contracted according to this ratio (see figure 4)

$$newlength_{cor} = oldlength_{cor} + (ratio_{change} * oldlength_{cor}).$$

So if a segment length is shortened, the side length is increased thus (approximately) preserving the area of the associated triangle.

6.2 Second Pass

In the second pass, the penetration check is done on the final equi-potential surface. Since penetrations with the skeleton were detected already, only few points of the triangular mesh representing the equi-potential surface are expected to penetrate the palate. These points are simply projected onto the palate.

7 Speech

Different tongue shapes differentiate phonemic elements. Consonants and vowels show different characteristics; for vowels, the entire tongue surface matters as well as its curvature while for consonant it is mainly the points of contact between the tongue and the palate or teeth that matter. For consonants, the tongue touches the palate with more tension than for vowels.

Jaw actions occur during accented vowel production. During jaw opening the tongue has greater distances to cover to reach its maxima positions. Depending on the speech rate, the tongue might not have time to reach these positions.

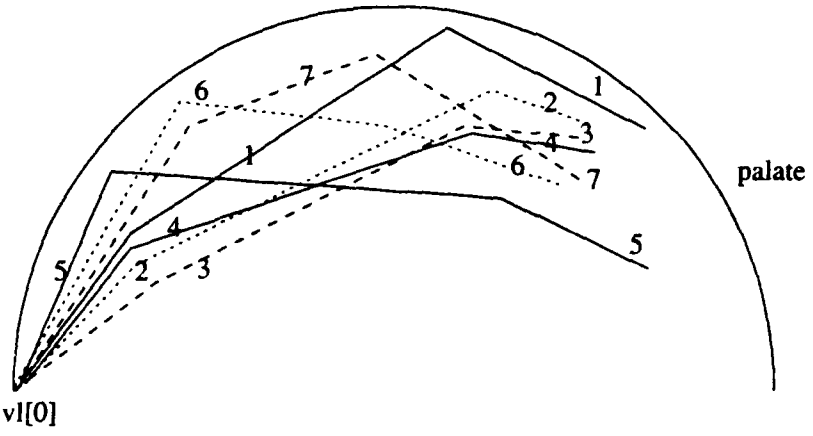
As it is noted in [10], there is not a universal tongue shape for each articulation, but the constraints on the tongue are such that to each articulation corresponds a particular shape which can appear in various positions in the oral cavity. The relevant issue here is the relation between the different tongue positions.

As speech rate increases the tongue does not have time to reach its extreme positions; the tongue shows less displacement, but its curvature is not affected by higher speed rates. Curvature is accentuated with loudness.

For slow speech-rate, steady-state tongue behavior occurs where the tongue remains still. [10] found coarticulation effects in the tongue motion during speech production. To compute lip shapes, our model uses a look-ahead model with some temporal and geometric constraints [18]. Our tongue model uses also the look-ahead model to compute the tongue shape.

8 Coarticulation

Using the tool we discussed in the previous sections, we specified a tongue shape for each vowel and consonant (see figure 5). Even though there is no universal shape for each phonemic item, we define one tongue shape to each phonemic item for the sake of simplicity [11].



1: high-front vowel such as 'heed'
2: high-mid-front vowel such as 'hid'
3: low-mid-front vowel such as 'head'
4: low-front vowel such as 'had'
5: low-back vowel such as 'father'
6: mid-back vowel such as 'good'
7: high-back vowel such as 'food'

Figure 5: Tongue shape during vowel production

Next, speech is decomposed into a sequence of discrete units such as syllables and phonemes. The lip shape and tongue shape of any given phoneme are influenced by its predecessors and successors due to a phenomenon called *coarticulation.*

8.1 FACS

Each facial expression is defined by a set of Action Units. An action unit is part of the Facial Action Coding System (**FACS**) developped by P. Ekman and W. Friesen. This system describes any visible facial action by the changes occurring beneath the muscular activity. An Action Unit (**AU**) corresponds to an action produced by one or more related muscles. Using such a notational system allows the computation of facial expression for a given animation to be independent of the chosen facial model. We refer the reader to [4] for a detailled description of each **AU**.

8.2 Computation of the Lip Shapes

In previous work ([18]) we implemented an algorithm computing the lip shapes. This algorithm is based on lip reading techniques. Vowels and consonants are divided into clusters corresponding to their associated lip shapes. Such clustering depends on the speech rate. The faster a person talks, the more marginally visible segments will lose their characteristic lip shapes.

The computation of the lip shapes is done in 3 steps:

- First, we apply coarticulation rules such as forward and backward rules derived from the look-ahead model. These rules deal with the fact that a segment may be influenced by a following or preceding vowel;
- Next, we look at temporal constraints where we consider the relaxation and contraction time of each **AU**. Indeed, we check that each **AU** has time to contract after the previous segment or relax before the next one. If not, the previous segment will be influenced by the contraction of the current segment and similarly for relaxation time.
- Finally, we look at geometric constraints by considering the surrounding phonemes. The intensity of an action is rescaled to take into account the geometric relation between successive segments. For example, when saying the word "popcorn", the 'o' of 'pop' is less open due to the 2 surrounding p's which are formed by the closure of the lips.

8.3 Computation of Tongue Shapes

We applied a similar look-ahead algorithm to compute the tongue shape. Phonemic segments show a slightly different clustering scheme for the tongue shapes in comparison with the lip shapes. The look-ahead model assures that for some highly deformable phonemic segments, the tongue shape will be influenced by the surrounding segments. If no tongue shape is associated to a particular segment, the program uses the property of the tongue which states that when a gesture is not involved in a particular segment but is in the next one, this gesture starts earlier [21]. In this case, the program starts the tongue movement on the previous segment which shows no tongue movement (e.g for /be/, the tongue associated to phoneme /e/ in not engaged in the production of /b/ and therefore starts as the same time as /b/ is pronounced [21]).

The program cutputs the different values of the tongue skeleton for each key-frame (each phonemic item correspond with a key-frame). The final tongue shapes are computed using the soft object program. The animation is displayed using *Jack*®a graphics package developed at the University of Pennsylvania.

9 Conclusion

We have presented a tool to model tongue movement during speech production. Tongue movement plays an important part in speech production. It helps to differentiate some phonemic elements when they can not be differentiated by their associated lip shapes. Tongue shape is very complex and flexible. The soft object technique allows to model a complex and flexible shape; at the same time these shapes are defined by few primitives which can be easily modified to create other shapes.

References

[1] R.A.W. Bladon and F.J. Nolan. A video-fluorographic investigation of tip and blade alveolars in english. *Journal of Phonetics*, 5:185–193, 1977.

[2] C.P. Browman and L. Goldstein. Gestural specification using dynamically-defined articulatory structures. *Journal of Phonetics*, 18:299–320, 1990.

[3] Michael M. Cohen and Dominic W. Massaro. Modeling coarticulation in synthetic visual speech. In D. Thalmann N. Magnenat-Thalmann, editor, *Computer Animation '93*. Springer-Verlag, 1993.

[4] P. Ekman and W. Friesen. *Facial Action Coding System*. Consulting Psychologists Press, Inc., 1978.

[5] J.W. Folkins, R.N. Linville, J.D. Garrett, and C.K. Brown. Interactions in the labial musculature during speech. *Journal of speech and hearing research*, 31:253–264, 1988.

[6] G. Heike, R. Greisbach, and B.J. Kroger. Coarticulation rules in an articulatory model. *Journal of Phonetics*, 19:465–471, 1991.

[7] Eric Keller. Factors underlying tongue articulation in speech. *Journal of Speech and Hearing Research*, 30:223–229, june 1987.

[8] R.D. Kent. Some considerations in the cinefluorographic analysis of tongue movements during speech. *Phonetica*, 26:16–32, 1972.

[9] R.D. Kent. *The Production of Speech*, chapter The Segmental Organization of Speech. Springler-Verlag, 1983.

[10] R.D. Kent and K.L. Moll. Tongue body articulation during vowel and diphthong gestures. *Folia Phoniatrica*, 24, 1972.

[11] P. Ladefoged. *A course in Phonetics*. Harcourt Brace Javanovich, 1982.

[12] J.P. Lewis and F.I. Parke. Automated lip-synch and speech synthesis for character animation. *CHI + GI*, pages 143–147, 1987.

[13] B. Lindblom. *The Production of Speech*, chapter Economy of Speech Gestures. Springler-Verlag, 1983.

[14] N. Magnenat-Thalmann and D. Thalmann. The direction of synthetic actors in the film *rendez-vous à montrèal*. *IEEE Computer Graphics and Applications*, pages 9–19, December 1987.

[15] S.E.G. Ohman. Coarticulation in vcv utterances: Spectrographic measurements. *Journal of Acoustical Society of America*, 39:151–168, 1966.

[16] S.E.G. Ohman. Numerical model of coarticulation. *Journal of Acoustical Society of America*, 41(2):311–321, 1967.

[17] F.I. Parke. Control parametrization for facial animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '91*, pages 3–14. Springer-Verlag, 1991.

[18] C. Pelachaud, N.I. Badler, and M. Steedman. Linguistic issues in facial animation. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation '91*, pages 15–30. Springer-Verlag, 1991.

[19] Elliot L. Salztman and Kevin G. Munhall. A dynamical approach to gestural patterning in speech production. *Ecological Psychology*, 1(4):333–382, 1989.

[20] M. Stone. A three-dimensional model of tongue movement based on ultrasound and x-ray microbean data. *Journal of Acoustical Society of America*, 87(5):2207–2217, 1990.

[21] M. Stone. Toward a model of three-dimensional tongue movement. *Journal of Phonetics*, 19:309–320, 1991.

[22] M. Stone, K.A. Morrish, B.C. Sonies, and T.H. Shawker. Tongue curvature: A model of shape during vowel production. *Folia Phoniatrica*, 39:302–315, 1987.

[23] M. Unser and M. Stone. Automated detection of the tongue surface in sequences of ultrasound images. *Journal of Acoustical Society of America*, 91(5):3001–3007, 1992.

[24] C.W.A.M. van Overveld and B. Wyvill. Potentials, polygons and penguins: An adaptive algorithm for triangulating and equi-potential surface. 1993.

[25] D.H. Whalen. Coarticulation is largely planned. *Journal of Phonetics*, 18:3–35, 1990.

[26] Sidney A.J. Wood. X-ray data on the temporal coordination of speech gestures. *Journal of Phonetics*, 19:281–292, 1991.

[27] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for Soft Objects. *The Visual Computer*, 2(4):227-234, April 1986.